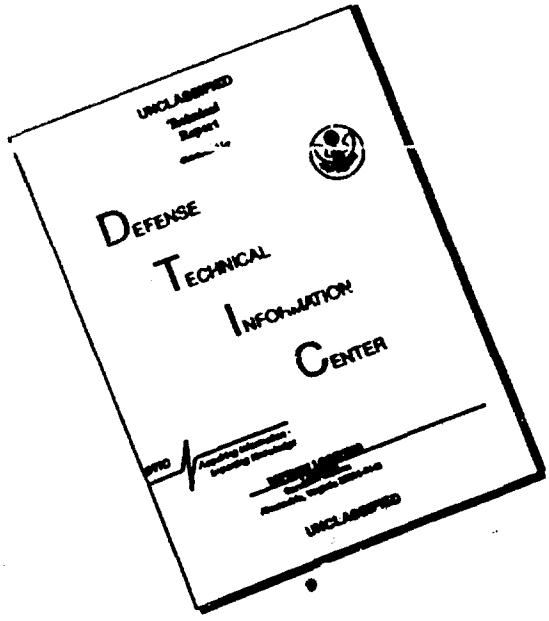
A Methodology for Optimal Planning over Time

Volume I—Main Report

by Charles A. Allen
Beverly D. Causey
James E. Falk
Ronald G. Magee
Charles W. Mylander
Ronald New, *Project Director*
John D. Pearson
Phillip D. Robers

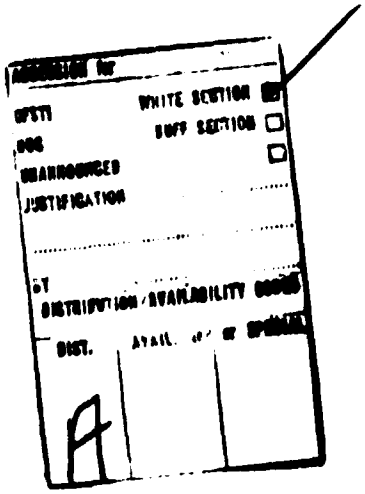
RAC

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.



UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) RESEARCH ANALYSIS CORPORATION	2a. REPORT SECURITY CLASSIFICATION none
	2b. GROUP N/A

3. REPORT TITLE

A Methodology for Optimal Planning Over Time, Volume I, Main Report

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Technical Paper

5. AUTHOR(S) (First name, middle initial, last name)

Charles A. Allen, Beverly D. Causey, James E. Falk, Ronald G. Magee, Ronald New, Project Director, John D. Pearson, Philip D. Robers, Charles W. Mylander

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
January 1972	86	5

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
DAHC19-69-C-0017 b. PROJECT NO. 011.310 c. d.	RAC-TP-445-Vol-I 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

"Approved for public release; distribution unlimited."

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	US Army, Combat Developments Command Combat Systems Group

13. ABSTRACT

This report describes a methodology which can be used to identify the most cost-effective plan for the phase-in and phase-out of vehicle systems--a methodology for optimal fleet planning over time. Volume I provides a systematic development of the problem structure, a qualitative description of the solution procedure, and mathematical and operational descriptions of the algorithm. Volume II provides appendices containing a demonstration problem, subroutine descriptions, program flow charts, program listings, and error message descriptions.

DD FORM 1 NOV 68 1473

UNCLASSIFIED

Security Classification

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
branch and bound						
fleet planning						
non-convex programming						
non-linear programming						
optimization over time						
vehicle systems						

Security Classification

SCIENCE AND TECHNOLOGY DEPARTMENT TECHNICAL PAPER RAC-TP-445 Published January 1972

A Methodology for Optimal Planning over Time

Volume I—Main Report

by

Charles A. Allen
Beverly D. Causey
James E. Falk
Ronald G. Magee
Charles W. Mylander
Ronald New, *Project Director*
John D. Pearson
Philip D. Robers

Research Analysis Corporation

McLean, Virginia 22101

RAC

Area Code 703
893-5900

DEPARTMENT OF THE ARMY
OFFICE OF THE CHIEF OF RESEARCH AND DEVELOPMENT
WASHINGTON, D.C. 20310

DARD-ARS

1. Volume I, "A Methodology for Optimal Planning Over Time" and Volume II, "Appendices A, B, C, D and E in Support of a Methodology for Optimal Planning Over Time - Volume I," were prepared by the Research Analysis Corporation for the Combat Systems Group, United States Army Combat Developments Command, and document RAC study 011.310, "Aircraft Systems Least Cost Phase-In." Copies of these reports are forwarded for your retention and use.

2. The methodology described in these volumes was developed to meet in part the need of the US Army to determine an optimal plan for phasing in new aircraft systems to meet its worldwide commitments yet remain within budgetary constraints. It provides to planners a tool for use in planning situations involving consideration of large numbers of alternative systems and combinations of tasks.

3. The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

FOR THE CHIEF OF RESEARCH AND DEVELOPMENT:

STANLEY R. MEEKEN
Colonel, GS
Chief, Studies and Analyses Division

Published January 1972
by
RESEARCH ANALYSIS CORPORATION
McLean, Virginia 22101

CONTENTS

VOLUME I

SUMMARY S-1

INTRODUCTION I-1

CHAPTER 1 - PROBLEM STRUCTURE 1-1

CHAPTER 2 - A QUALITATIVE DESCRIPTION OF BRANCH AND BOUND 2-1

CHAPTER 3 - MATHEMATICAL DESCRIPTION OF PROBLEM AND ITS SOLUTION 3-1

THE CONSTRAINT SET 3-1

Materiel Balance Constraints - Consistency Constraints - Vintage Constraints - Master Variable Relation Constraints - Cost Constraints

THE OBJECTIVE FUNCTION 3-6

Primary Cost Categories - Secondary Cost Categories

THE SOLUTION PROCEDURE 3-10

Linear Envelopes - Bounding the Solution - Partitioning into Subsets - The Algorithm

CHAPTER 4 - AN OPERATIONAL DESCRIPTION OF THE BRANCH AND BOUND SOLUTION PROCEDURE 4-1

PROGRAM INTERFACING 4-1

THE MATRIX GENERATOR 4-5

Program Logic - Core Allocation - User's Subroutine (YRCOST) - Input Formats - Output Description

THE MAIN PROGRAM 4-21

Program Logic - Core Allocation - User's Subroutine (GETPHI) - Input Formats - Output Description

THE REPORT GENERATOR 4-34

Program Logic - Core Allocation - User's Subroutine (YRCOST) - Input Formats - Output Description

REFERENCES R-1

FIGURES: VOLUME I

1-1 A Typical List of Alternative Vehicle Arrays Which Could Service a Mission Group 1-1

1-2 Eight Independent Groups Each with Twenty Equally-Effective Alternatives 1-3

1-3 The Mission Group Tableau 1-5

FIGURES: VOLUME I (continued)

2-1	A Concave Cost Function	2-1
2-2	Partitioning the Total Solution Set Into Subsets	2-3
3-1	Cost Function	3-11
3-2	A Linear Envelope of a Concave Function with Discontinuity at Origin	3-13
3-3	The Algorithm	3-17
4-1a	Control Cards to Execute All Programs as a Single Job	4-3
4-1b	Control Cards to Execute Programs as Separate Jobs	4-4
4-2	System Macro Flowchart	4-6
4-3	Symbolic Naming Convention	4-7
4-4	Positive Integer Code	4-8
4-5	Basic Cards	4-13
4-6	Vehicle Table	4-14
4-7	Period Table	4-15
4-8	Task Table	4-16
4-9	Deck Structure for Matrix Generator	4-20
4-10	Input Data Formats	4-27
4-11	Deck Structure for Main Program	4-30
4-12	Deck Structure for Report Generator	4-37

CONTENTS
VOLUME II

APPENDIX A - SAMPLE PROBLEM	A-1
APPENDIX B - SUBROUTINE DESCRIPTIONS	B-1
APPENDIX C - PROGRAM FLOWCHARTS	C-1
APPENDIX D - PROGRAM LISTING	D-1
APPENDIX E - ERROR MESSAGES	E-1
GLOSSARY	G-1
REFERENCES	R-1

FIGURES: VOLUME II

A-1	Alternative Fleet Mixes for 1972	A-1
A-2	Capital Equipment (Truck) Requirements as a Function of Time	A-2
A-3	The Simplified Objective (cost) Function for the Sample Truck Problem	A-4
A-4	The Sample Problem Data Deck for the GENLCP Program	A-8
A-5	GENLCP Output For Sample Problem, Parts (a) - (f)	A-10
A-6	The Sample Problem Data Deck For the BBCAV2 Program	A-18
A-7	A Branching Tree For the Sample Problem	A-19
A-8	The Sample Problem Data Deck For the REPGEN Program	A-20
A-9	REPGEN Output For Sample Problem, Parts (a) and (b)	A-21
A-10	A Sample Optimal Plan For the Ace Trucking Company	A-24

SUMMARY

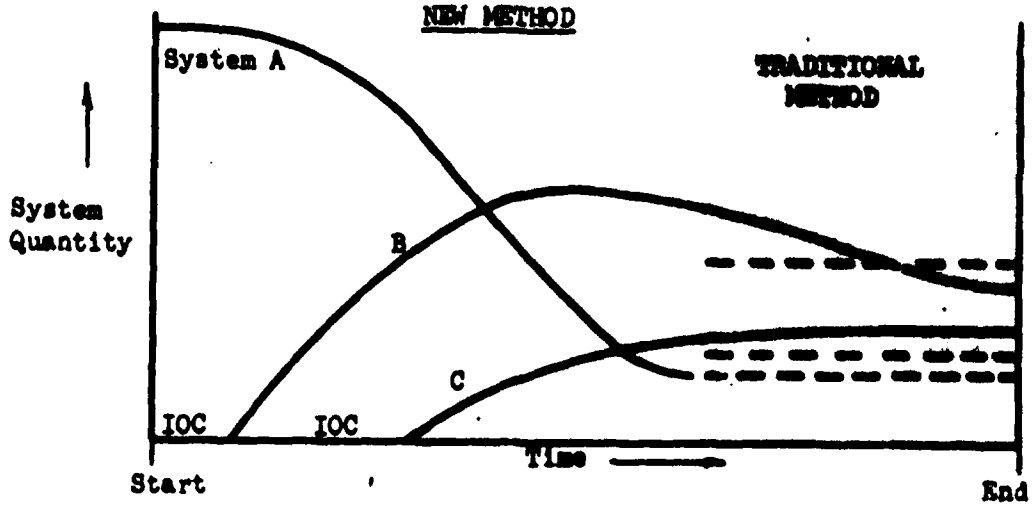
PROBLEM

To develop and demonstrate an algorithm (scheme) that will identify, from among the myriad of possible plans, the most cost-effective plan for the phase-in and phase-out of vehicle systems — a methodology for optimal fleet planning over time.

DISCUSSION

This report describes an optimization methodology that accepts as input the numbers and costs associated with alternative mixes of vehicles (machines), such that each mix can perform a series of tasks with equal effectiveness, and provides as output the numbers and cost of the "least-cost-mix" that will meet a prescribed level of effectiveness.

The concept of the new algorithm is easily seen by comparison with traditional methods of solutions as implied by the following sketch:



In the traditional methods, a future time (planning) period is selected, at the beginning of which all of the considered systems could be made available if desired, and certain approximations made regarding the inherited assets (existing systems) and their status. The optimization is then conducted on a one-time (snap-shot) basis for the selected period of time (typically 10-12 years) as implied by the dashed lines in the shaded portion of the sketch. No information is made available, or is provided, regarding the interim period between the present time and the start of the planning period. In addition, it is typical for all data (both input and output) to be considered to be averaged over the planning period.

In the new algorithm, the optimization is conducted "dynamically" over an extended period of time (say 15-25 years) beginning with the present status of the existing systems and concluding at the end of the planning period. This permits full and explicit consideration of the inherited assets and their aging characteristics as well as the initial operational capability (IOC) dates of the proposed new systems and their growth characteristics. The optimization is conducted over the complete (extended) period, phasing out old systems (e.g., system A) as obsolescence occurs, and phasing in new systems (e.g., systems B & C) as they become available. The response to changing military requirements is controlled by structuring the complete period in terms of smaller and smaller subperiods so as to obtain (almost) a continuous adjustment of the status of all systems, as implied by the solid lines in the sketch. It is noted with emphasis that the user must specify the input data array which details the task (mission group) requirements for each subperiod within the planning horizon.

In specific terms, the methodology:

- Provides a global (optimal) solution over the entire time period of interest,
- Accounts for, and makes optimal use of, inherited assets,
- Determines when and if to phase-in new systems,
- Determines if and when to phase-out existing systems,
- Can account for step-discontinuities in cost, such as development expenses,
- Can account for decreased procurement costs as the quantity of systems increases (learning curve effect),
- Allows input of increased operating costs, if appropriate, as the age of the vehicles increase,
- Allows input of budgetary constraints, by category if appropriate, and
- Accounts for "retained value" of the systems at the end of the time period.

The mathematical statement of the programming problem is of the following form: Find a vector $x = (x_1, \ldots, x_n)$ that will

$$\text{minimize } \phi(x) = \sum_{i=1}^{n} \phi_i(x_i)$$

subject to $x \in G$, $0 \leq x \leq L$

where

ϕ = the total cost,

x = a vector representing the number of vehicles of each type,

G = the constraint set,

L = the vector of upper bounds,

and the subscript i, i = 1, 2, ... n, denotes the vehicle type. There are two restrictions on the form of the cost function. First, the function must be "separable" which means that each type of vehicle

can be costed independently. The second restriction is that the cumulative cost curve for an increasing quantity of each type of vehicle must be concave. This means that a straight line connecting any two points on the curve lies on or below the curve at all intermediate points. These requirements are generally satisfied in cost minimization problems — that is, those problems where it is appropriate to minimize cost for a specified level of effectiveness. In addition, there are three restrictions on the constraint set: (1) all of the constraints must be linear, (2) none of the constraints can be strict inequalities, and (3) the number of vehicles of each type must be bounded from above and below.

The general procedure of the algorithm is as follows: The set of all possible solutions is successively broken into smaller and smaller subsets. For each such subset of solutions, a lower bound on the cost of the best solution in the subset is calculated. At each iteration of the algorithm the subset with the lowest lower bound is broken into two smaller subsets. Eventually, a subset consisting of exactly one solution is found and it has an actual cost that is less than or equal to the lower bound for all other subsets. This is the least cost solution. The lower bounds are found by solving the problem with linear cost functions instead of the actual concave cost functions.

Once the least-cost solution has been obtained, the sensitivity of this "optimal plan" to possible errors in the user specification of the input requirements can be determined. In other words, the consequences of the uncertainty of the input data can be explored by manipulation (calculations) within the computer program itself. Many of these sensitivity results can be displayed automatically, as standard output — other results can be efficiently re-computed from the results of the optimal plan.

Volume I of this report provides a systematic development of the problem structure, a qualitative description of the solution procedure, and mathematical and operational descriptions of the algorithm. Volume II provides appendices containing a demonstration problem, subroutine descriptions, program flow charts, program listings, and error message descriptions. Those readers who are interested in a general (non-mathematical) understanding of the methodology should confine themselves to the Introduction, Chapters 1 and 2, and possibly Appendix A. Those who are interested in the mathematical substance should concentrate further on Chapter 3. Chapter 4 and the remaining appendices are for the "users" and "programmers."

CONCLUSIONS

This algorithm provides Army planners with a comprehensive non-linear methodology which can be used for minimizing costs over long planning periods. While the emphasis in this report is on "fleet planning" exercises, the methodology is generally applicable to problem involving "machines" of any type (e.g., computers, motor/generator sets, radio sets, etc.).

A principal strength of the algorithm is the ease with which sensitivity analyses can be conducted and the facility with which input data may be changed to periodically update the planning process. The structure of the methodology is sufficiently general to be used in all optimization problems appropriate to the assumptions and conventions. The algorithm can accept large enough problems, and runs with sufficient rapidity to be used as a standard research tool.

INTRODUCTION

The set of criteria which one uses to select one capital investment proposal over another may vary, but there is no substitute for being able to view interactive business operations in their entirety; that is, it is certainly desirable to see the influence of an action, taken at a given time and place, on the overall costs (or effectiveness) of a major procurement program. This report is the result of such an effort in that most quantifiable factors which affect, or are affected by, a single capital equipment investment are aggregated and viewed in their interactive state. In this way, the program manager can test and probe a model of his operation to see the influence of each capital investment proposal.

The manager today is very often faced with decisions regarding the purchase of capital equipment within a very complex problem environment. These questions may be familiar ones—Should the old vehicles be replaced?...When?—But the new models won't be available until late in '73; should we order immediately?—That looks good now, but what about costs over the long term? These questions are certainly not new to the program manager—he has been "solving" them for years using experienced judgment and various rules-of-thumb; but, it is no secret that these methods are not altogether satisfactory. In fact, such methods are weak because they only begin to solve the problem...such methods are superficial in the sense that they appear to account for the primary factors, but fail because they do not consider the important interactions. The methodology reported on here utilizes an interactive model wherein

an account is taken of variations in the problem environment over time; i.e., the "dynamics" of the long term plan are inherent in the model.

In most cost-effectiveness analyses it is usual that one seeks to find an optimal solution — i.e., a minimum cost or a maximum effectiveness solution, subject to certain problem constraints. The following describes an optimization methodology which accepts as input the numbers and costs associated with alternative mixes of vehicles (machines), such that each mix can perform a series of tasks with equal effectiveness, and provides as output the numbers, schedule, and cost of the "least-cost-mix" that will meet the prescribed level of effectiveness. The problem that we consider here is dynamic, hence, we call this a "phasing" problem since its solution specifies the orderly and efficient phase-in and phase-out of a fleet of vehicles or machines. In this context, Hetrick[1] has noted that: "Some managements pay only lip service to the idea that decisions should be made to the best advantage of the total enterprise and for the long term. All too frequently, short term decisions are made that are crippling in the long term." It may be convenient to think of this problem as a multi-stage decision process where the interdependence between stages (sub-periods) is due to the inheritance of a fleet of machines from a previous sub-period.

The algorithm that is used to obtain the least-cost solution is of the "branch-and-bound" type and solves a sequence of sub-problems, each of which evolves into a linear program.[2]

In Chapter 1 of Volume I we structure the problem by systematically developing the array of alternatives to satisfy prescribed requirements.

Here the requirements are aggregated into independent groups of missions in building-block fashion and a concise problem statement is developed. Chapter 2 provides a qualitative description of the solution procedure in order to demonstrate its logic and intuitive appeal. A mathematical description of the algorithm is given in Chapter 3; here the problem statement is transformed into the general non-linear programming form, and the solution procedure explained with some rigor. Chapter 4 is an operational description that translates the algorithm into the language used in the computer code.

Volume II contains a series of appendices (A through E) designed for the user and programmer. Included are a detailed description of a sample problem, subroutine descriptions, program flowcharts, a program listing, and an explanation of all program error messages.

Chapter 1

PROBLEM STRUCTURE

Let us assume that there exist or that one can generate, lists of alternative combinations of vehicles, such that each alternative satisfies the requirements of a particular mission group.* For example, a typical set of these alternatives might look like that in Fig. 1.

Vehicle Type / Alternative #	Vehicle #1	Vehicle #2	Vehicle #3	Vehicle #4	Vehicle #5
1	46	15	63	104	138
2	0	15	82	104	138
3	46	0	70	104	138
4	0	0	90	104	138
5	46	15	0	167	138
6	0	15	0	187	138
7	46	0	0	176	138
8	0	0	0	196	138
9	46	15	63	0	265
10	0	15	82	0	265
11	46	0	70	0	26[illegible]
12	0	0	90	0	265
13	46	15	0	0	313
14	0	15	0	0	328
15	46	0	0	0	315
16	0	0	0	0	332

Fig. 1-1. A Typical List of Alternative Vehicle Arrays Which Could Service a Mission Group

Ideally, each of the alternatives (rows) on a given list must satisfy the requirements of the corresponding mission group with equal effectiveness.

* "Mission group" is meant to be synonymous with "series of tasks" or "collection of jobs"; the important point here is that vehicles (machines) may be "pooled" to accomplish the objective(s), i.e., each "job" need not be performed by physically different vehicles.

RAC

The job of the optimization methodology is to choose one alternative from each list (mission group) so that the total mix is least costly. For a single mission group the task here is a simple one, i.e., one need merely evaluate the cost of each and every alternative, and then select that one having the least cost. As long as this list is not unreasonably long (and a list of one million alternatives is not unreasonably long), then the task is not prohibitive. When there exists two or more independent* mission groups, then the number of feasible combinations goes up rather quickly. Consider, for example, a problem involving eight independent mission groups each of which contains twenty alternatives, as shown in Fig. 1-2. Since the cost relationships are in general non-linear (fixed R&D charges and learning curves), the least-cost-mix will not be the simple combination of the least-cost alternative for each mission group. The optimization problem here is to select one alternative from each of the eight mission groups such that this combination (mix) of alternatives is the least-cost-mix. Again, we could attempt to explicitly enumerate each and every combination, but a little thought will reveal that there exist 20^8 possible combinations. If each combination could be evaluated in one-hundredth of a second on a modern computing machine, the total set of evaluations would require more than 50,000 hours of computing time. It is this problem (and only this problem) which precludes explicit enumeration of all possible solutions and requires the use of a more sophisticated optimization technique. Furthermore, we have not yet considered the additional complexity introduced by the element of time, i.e., the problem is not yet dynamic. The requirements represented by the eight independent mission groups described above may be

* Mission groups are independent if the vehicles used to satisfy one mission group are not (cannot be) used to satisfy the other.

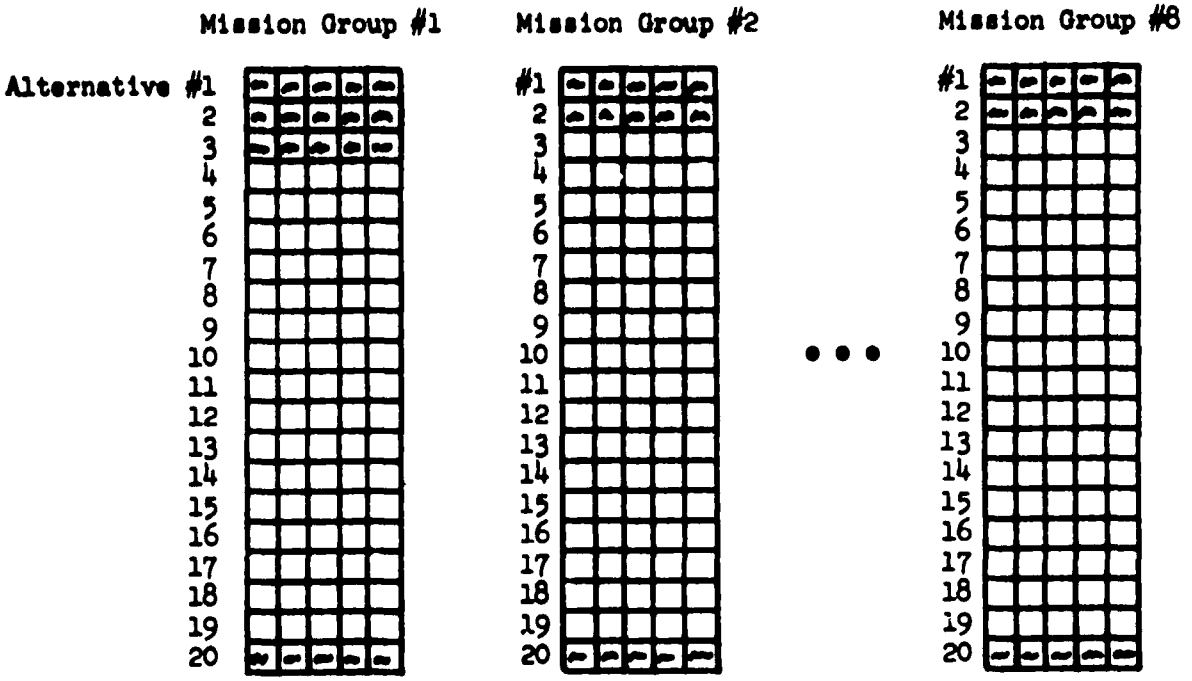


Fig. 1-2—Eight Independent Mission Groups Each with Twenty Equally-Effective Alternatives

thought of as originating at eight separate geographical locations but occurring at essentially the same time — it is in this sense that the mission groups are independent. Actually we would like to consider the more general problem of satisfying an array of requirements in a sequence of time periods commencing now and ending at sometime in the future. We can accommodate this additional time dimension in the problem by developing arrays of mission groups like those of Fig. 1-2 for each sub-period within the total time period of concern. To continue with our example, we could develop a tableau like that of Fig. 1-3 which displays eight mission groups for each of 15 yearly sub-periods from 1971 thru 1985. This is not to imply that there must exist any regularity or uniformity to the number of mission groups per sub-period, or alternatives per mission group, etc. — there need only exist a finite number of equally effective alternatives for each independent mission group within each sub-period. Of course, the duration of the sub-period(s) is a measure of the "resolution" of the problem and should be selected by the model manipulator (user) from the state of knowledge which exists of the predicted requirements for that sub-period.

Finally, we complete the description of the problem structure by noting that there can exist at the beginning of the first sub-period an inherited fleet of machines. This inherited fleet is characterized by number, type, and vintage* and may be used to satisfy part or all of the requirements of subsequent sub-periods. It may now be obvious that the interdependency between sub-periods derives from the inheritance of machines from sub-period to sub-period and if that inheritance

*The vintage or age of each inherited machine will, in general, influence the operating cost and salvage value of that machine.

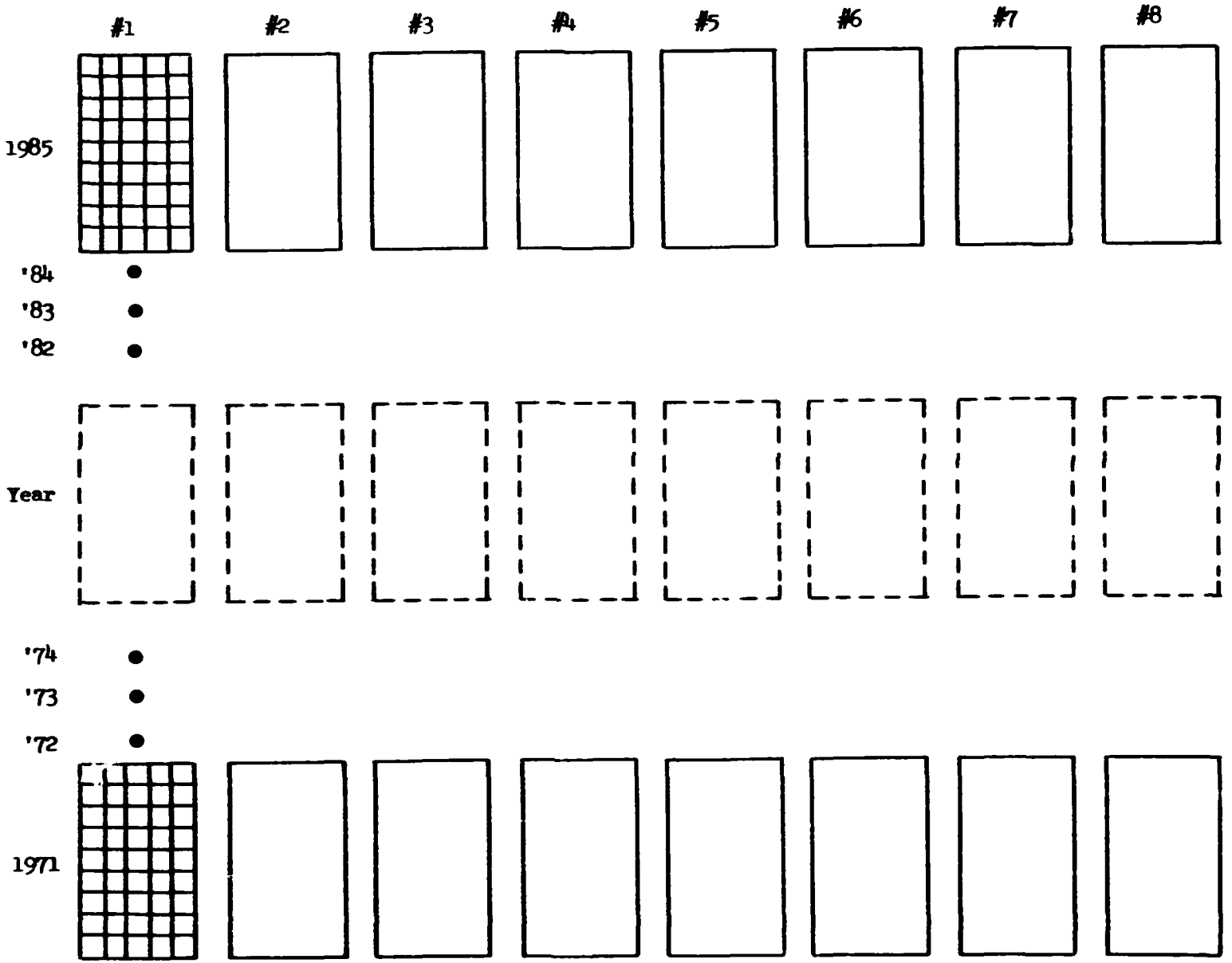


Fig. 1-3—The Mission Group Tableau

were ignored, then the least-cost-solution for the entire time period would be the sum of the least-cost-solutions for each sub-period.

The problem structure as defined in the preceding paragraphs provides the foundation for a concise problem statement:

PROBLEM STATEMENT

We seek to satisfy the fixed and prescribed requirements in each and every sub-period with that mix of machines which will minimize the total overall cost. This least-cost-mix must obtain from the selection of one alternative for <u>each</u> mission group within <u>each</u> sub-period, in conjunction with an optimal retention of inherited assets.

Chapter 2

A QUALITATIVE DESCRIPTION OF THE BRANCH-AND-BOUND SOLUTION PROCEDURE

The number of possible (feasible) solutions to a typical "multi-stage phasing" problem is enormous; the example problem represented by the tableau in Fig.1-3has more than 20^{120} possible solutions! Clearly, some systematic solution procedure which by-passes complete enumeration of all possible solutions is needed.

We seek to minimize some cost (objective) function which, at the very least, reflects the expense involved in developing, purchasing, and operating a fleet of machines over an extended time period. Furthermore, this minimization must be carried out subject to certain requirement, production, and cost constraints. The specific details of the cost equation can be deferred for the moment, but the general form of the function is important here. A typical vehicle cost curve, plotting total cost versus number of vehicles purchased and operated is shown in Fig. 2-1 below.

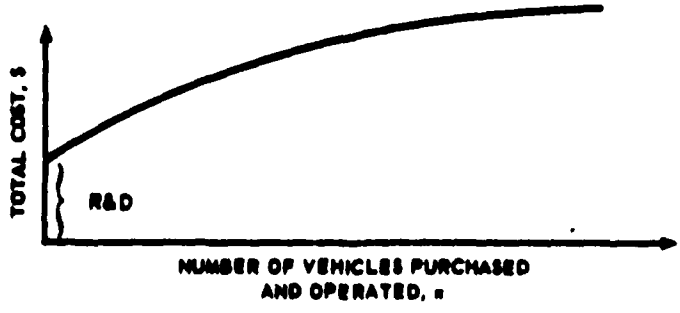


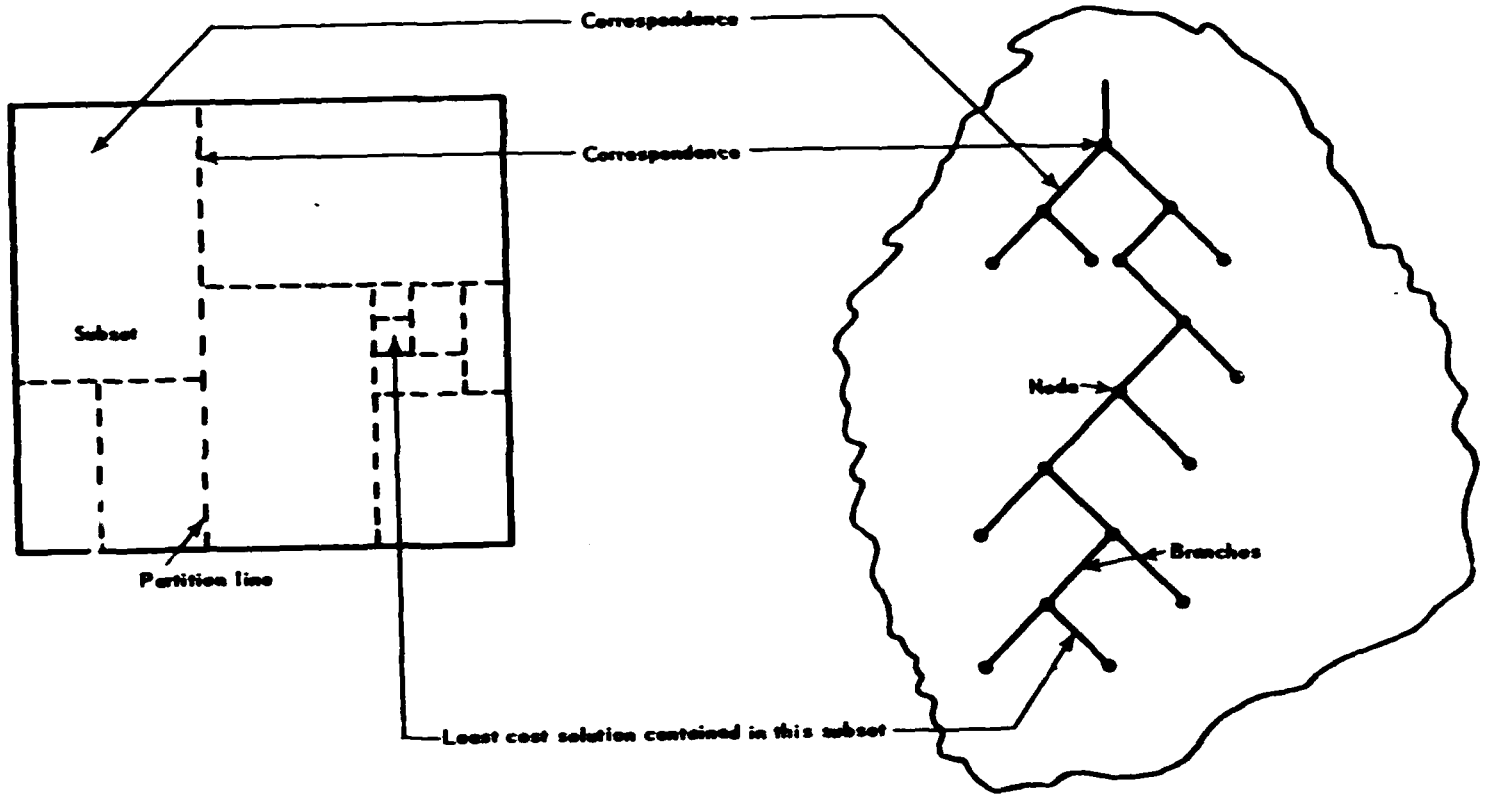
Fig. 2-1—A Concave Cost Function

The cost jump at x = 0 represents the R&D expenditure, which must occur prior to the procurement of any number of this particular vehicle. For x greater than 0, the deviation from linearity derives from the "learning" advantage of a volume purchase. The function represented by this curve is called a "concave" function and it is this characteristic which motivates the branch-and-bound solution procedure. A concave function of a single variable is one whose graph is never "below" a straight line joining any two points of that function.* That is, the straight line value is always less than or equal to the function value and we note, importantly, that this straight line approximation will always yield an "underestimate" of the concave cost function.

Now, let us visualize enclosing all 20^{120} solutions of our sample problem by the solid line in Fig. 2-2a. This box-like enclosure contains the entire set of solutions and we will ultimately partition this set into subsets, the partition being indicated by the dotted lines. A second representation of the partitioning procedure is shown by the "branching" diagram in Fig. 2-2b; here, a tree of branches is formed in one-to-one correspondence with the partitioned subsets with each node corresponding to a dotted partition line. Each node on the branching tree identifies a linear programming problem. Initially, we obtain the cost of one (any one) mix; this cost is called the reference solution. We divide the total set of solutions into two subsets, i.e., we form two branches (see diagram). We then calculate a lower bound** (underestimate) to the solutions in each subset by solving the corresponding

* A more general and more rigorous definition of a concave function is given in Chapter 3.

** A lower bound in a given subset is a cost less than or equal to the cost of any solution in that subset.



a. A Symbolic Enclosure Containing All Solutions

b. The Branching Tree

Fig. 2-2—Partitioning the Total Solution Set into Subsets

linear programs and compare these lower bounds to the cost of the reference solution. There are three cases to consider: (1) both lower bounds are less than the cost of the reference solution, (2) one lower bound is less than the reference solution, (3) neither of the lower bounds are less than the cost of the reference solution. A description of the procedure to be followed for case (2) will suffice for all cases. If a lower bound is greater or equal to the reference solution, its corresponding subset <u>cannot</u> contain a better solution than the reference solution and this is the basis for discarding this entire subset of solutions. If a lower bound is less than the reference solution, the corresponding subset <u>can</u> contain the least-cost-solution; hence, this subset is partitioned again and the testing process repeated. For each subset having a lower bound less than the current reference solution, a new reference solution is calculated from this candidate subset. If this new solution is less than the old reference solution, we replace the old with the new. This entire process can be shown to converge to the least-cost-solution of the entire set of solutions. The specific details as to how one calculates the lower bounds, partitions the subsets, chooses the branching rules, etc., is left for Chapter 3. Nevertheless, it should now be clear that the efficiency of this algorithm stems from the freedom to <u>legitimately</u> discard large groups of possible solutions without ever explicitly evaluating them — only a small fraction of the total number of solutions need be evaluated.

Finally, it should be noted that the branch-and-bound procedure actually considers many more than the 20^{120} (or so) discrete possible

solutions to our sample problem. In fact, a <u>continuum</u> of possible solutions is considered, comprised of the 20^{120} discrete solutions plus <u>all</u> convex* combinations of the discrete solutions. In so doing, it is assumed that all such convex combinations represent equally effective alternative solutions to the problem.

---

* A "convex" combination is a linear combination of a special type, e.g., a convex combination of two vectors $\bar{U}_1$ and $\bar{U}_2$ is $[\alpha\bar{U}_1 + \beta\bar{U}_2]$ such that $\alpha + \beta = 1$.

Chapter 3

MATHEMATICAL DESCRIPTION OF THE PROBLEM AND ITS SOLUTION

The mathematical problem statement is of the following form:

Find a vector $x = (x_1, \ldots, x_n)$ which will

$$\text{minimize } \Phi(x) = \sum_{i=1}^{n} \Phi_i(x_i)$$

$$\text{subject to } x \in G,$$

$$0 \leq x \leq L$$

We develop now the specific and detailed structure of the model beginning with the constraint set, G.

CONSTRAINT SET

In this problem there are three basic activities which interact, and associated with each activity is a problem variable. The value of the problem variable indicates the level at which the activity is to be conducted. The first activity is the disposition of the inherited assets. The problem variable is $w_{j\ell m}$, which refers to the number of j^{th} type machines which were purchased at the beginning of subperiod ℓ and will be disposed of at the end of subperiod m. The second activity is the purchase and disposition of new machines, and its variable is $x_{j\ell m}$, where the subscripts have the same meaning as above. The third activity is related to the performance of the mission requirements. The problem variable is $p_{ik\ell}$, the fraction of alternative k to be used in accomplishing the i^{th} mission group in subperiod ℓ.

Materiel Balance Constraints

The most basic constraints relate the interaction between activities. They insure that the machines which we have available in any subperiod are sufficient to meet the mission requirements of that subperiod. The machines available in subperiod ℓ are determined by adding those remaining from the inherited assets to those which exist as a result of new purchases, that is,

$$\sum_{\ell'=K_j}^{0} \sum_{m=\ell}^{Y} w_{j\ell'm} v_{\ell-\ell',j} + \sum_{\ell'=1}^{\ell} \sum_{m=\ell}^{Y} x_{j\ell'm} v_{\ell-\ell',j}$$

where K_j = number of subperiods before the start of the planning period from which the j^{th} type of machine is inherited. Note that ($\ell' = K_j,\ldots,-1,0$) in the first series set above but ($\ell' = 1,2,\ldots\ell$) in the second series set. Y = number of subperiods in the planning horizon ($\ell = 1,2,\ldots,Y$). The factor $v_{\ell-\ell',j}$ accounts for attrition due to accidental loss of machines and specifically represents that fraction of type j machines which remain usable after $\ell-\ell'$ subperiods of existence.

The number of the j^{th} type machines required in subperiod ℓ is obtained by summing over all units used to satisfy each mission group. This summation is expressed as

$$\sum_{i\in M_\ell} t_{i\ell} \sum_{k=1}^{R_{i\ell}} u_{ijk\ell} p_{ik\ell}$$

where M_ℓ is the set of different mission groups in subperiod ℓ, $R_{i\ell}$ is the number of alternative ways of performing mission group i in

RAC

subperiod ℓ, $t_{i\ell}$ is the number of times mission group i must be independently performed in subperiod ℓ, and $u_{ijk\ell}$ is the number of units of the j^{th} equipment used in performing mission group i with the k^{th} alternative in subperiod ℓ.

The constraint relation insures that the number of j^{th} type machines that we have on hand for use in subperiod ℓ must be greater than or equal to the number required. Expressed as an equality, this relation is:

$$\sum_{\ell=K_j}^{0} \sum_{m=\ell}^{Y} w_{j\ell m} v_{\ell-\ell',j} + \sum_{\ell=1}^{\ell} \sum_{m=\ell}^{Y} x_{j\ell m} v_{\ell-\ell',j} = \sum_{i \in M_\ell} t_{i\ell} \sum_{k=1}^{R_{i\ell}} u_{ijk\ell} + s_{j\ell}$$

$$\text{for } j = 1,2,\ldots,N \text{ and } \ell = 1,2,\ldots,Y$$

where $s_{j\ell}$ is a slack variable which is specifically included because it has a physical meaning in the problem; it represents machines of the j^{th} type which can be "mothballed" in subperiod ℓ for later use.

Consistency Constraints

The second subset of constraints insures that the requirements of each mission group are met. This is accomplished by requiring that the fractions or proportions used for each of the alternatives within a given mission group do sum to 1.0. The constraint set then takes the form:

$$\sum_{k=1}^{R_{i\ell}} p_{ik\ell} = 1$$

$$\text{for } \ell = 1,2,\ldots,Y \text{ and } i = 1,2,\ldots,M_\ell$$

It should be emphasized that the formulation of the consistency constraints in this manner actually represents the assumption that all convex combinations of alternatives are equally effective. This assumption introduces two points of concern: (1) In most cases this assumption will result in a fractional number of machines in the optimal solution, requiring the user to readjust the solution to the nearest "whole" machine(s)—this interpretation of fractional solutions will cause no large errors as long as the number of machines of each type is large, relative to one; and (2) this assumption may result in the choice of a particular convex combination which differs significantly from that of an equally effective alternative. The seriousness of this problem can only be determined by a study of a series of typical problems. Until such further study is conducted, it can only be noted that, to date, no serious departures from equal effectiveness have been observed in the problems which have been solved.

Vintage Constraints

The next set of constraint relations accomplishes two purposes; (1) they define the inherited assets in terms of both number and vintage (age) and, (2) they relate these numbers to the ultimate disposition of the assets to ensure that all are accounted for. These relations are given by the equations:

$$W_{j\ell} = \sum_{m=0}^{L_j-1+\ell} w_{j\ell m} \quad ; \quad \begin{matrix} j = 1,2,\ldots,N \\ \ell = K_j, K_j+1, \ldots, -1, 0 \end{matrix}$$

where L_j is the maximum life (in subperiods) of machine type j, and $W_{j\ell}$ is the number of type j machines inherited from subperiod ℓ.

Master Variable Relation Constraints

We define x_j to be the total number of machines of type j purchased during the entire planning period. We can obtain x_j by summing the subordinate variables $x_{j\ell m}$ over all ℓ and m, the purchase and disposal subperiods.

$$x_j = \sum_{\ell=1}^{Y} \sum_{m=\ell}^{Y} x_{j\ell m} \; ; \quad j = 1,2,\ldots, N$$

These constraints not only define the master variables but also maintain "separability" in the cost function as discussed on page 3-10.

Cost Constraints

Specifically at this point we discuss and describe the implementation of linear procurement constraints. Similar constraints could be considered for operating costs but their implementation would require program changes. This constraint restricts the total purchase in subperiod ℓ to be less than or equal to some fixed ceiling, H_ℓ, for procurement in that subperiod. The constraint relation allows for any funds left over at the end of subperiod ℓ to be transferred for use in subperiod $\ell + 1$. This is given as

$$H_\ell = \sum_{j=1}^{N} a^{o}_{j} \sum_{m=\ell}^{Y} x_{j\ell m} + P_\ell - P_{\ell-1}$$

$$\text{for } \ell = 1,2,\ldots,Y$$

where P_ℓ is the slack variable representing the surplus funds, and a^{o}_{j} is the linear cost coefficient (which may represent an approximation if the cost function is nonlinear). This constraint set is intended to be incorporated as an optional feature since its use may very well result in an infeasible problem.

THE OBJECTIVE FUNCTION

Six distinct cost categories are included within the objective function. The three primary categories are research and development (R&D), procurement, and operating expense. The secondary categories are most appropriately named mothballing, salvage, and truncation costs (these terms will be defined in a subsequent paragraph). For purposes of this development, only the R&D and procurement cost functions are considered to be concave; however, from a modeling point of view there is no reason why any of the other categories could not also be defined by concave functions.

Primary Cost Categories

The R&D function can be represented by a step discontinuity at zero. We will designate this function as $U_j(x_j)$. Then $U_j(x_j) = 0$ if the master variable $x_j = 0$, and $U_j(x_j) = D_j$ (some constant) if $x_j > 0$. D_j is the estimated R&D cost for machine type j. The total R&D cost is obtained by summing over all machine types,

$$\sum_{j=1}^{N} U_j(x_j)$$

and this is the first type of term in the objective function.

Procurement is defined in terms of a so-called "learning curve" and can in most cases, be represented by the function $a_j x_j^{b_j}$; where a_j is the cost of buying one machine of type j, and $0 < b_j \leq 1$. This function suggests that although the total cost increases as the buy size increases, the unit cost is actually decreasing (except for the linear case when $b_j = 1$). The total expenditure for procurement is then,

$$\sum_{j=1}^{N} a_j x_j^{b_j}$$

Since we distinguish between the inherited and purchased machines, it is necessary to develop two separate expressions for operating cost. We define for use in both expressions the cost coefficient c_{jk}, i.e., the cost of operating one unit of the j^{th} type in its k^{th} subperiod of existence. This allows for an increase in operating cost based on the age of the equipment.

Starting with the new equipment, we wish to formulate the operating costs in terms of variables of the type $x_{j\ell m}$. Recall that $x_{j\ell m}$ represents the number of units purchased at the beginning of subperiod ℓ and disposed of at the end of subperiod m. Then the lifetime of $x_{j\ell m}$ machines is $m - \ell + 1$ subperiods, and the total operating cost for these machines is the sum of the costs for each subperiod, that is:

$$\sum_{k=1}^{m-\ell+1} c_{jk}\, x_{j\ell m}$$

By summing over all subperiods and machine types, we derive the total operating cost for the new machines over the entire planning period. This sum is

$$\sum_{j=1}^{N} \sum_{\ell=1}^{Y} \sum_{m=\ell}^{Y} \sum_{k=1}^{m-\ell+1} c_{jk}\, x_{j\ell m}$$

The operating cost of the inherited assets is calculated similarly. The cost of a given variable, $w_{j\ell m}$, is

$$\sum_{k=2-\ell}^{m-\ell+1} c_{jk}\, w_{j\ell m}$$

where the summation begins at $2 - \ell$ since ℓ here is negative or zero and the first subperiod of the planning period must be at least the second

subperiod of the machines life. The total operating cost is again obtained by summing over the subperiods and machine types; that is,

$$\sum_{j=1}^{N} \sum_{\ell=K_j}^{O} \sum_{m=1}^{Y} \sum_{k=2-\ell}^{m-\ell+1} c_{jk} w_{j\ell m}$$

Secondary Cost Categories

Each of the primary categories represents a specific cost (outlay of dollars). In contrast, each of the secondary categories represents a type of savings or credit to the system. "Mothballing" is herein defined as the storage of equipment (owned but not presently needed) for use at a later time. In the present formulation, we associate a savings with this storage resulting from a decrease in operating cost (i.e., we initially charged the full operating cost for all retained machines). "Salvage" is the refund received from the sale of equipment which is no longer needed. Finally, "truncation" is a credit for the value of the equipment on hand at the end of the planning period. It is used to compensate for the fact that we only consider a finite time period and have no real knowledge concerning the use of the equipment after that time. In the current formulation of the model we have assumed that each of these three secondary cost categories can be approximated by linear cost expressions--this is not meant to imply a restriction; more complex cost expressions may be used.

The mothballing variable, $s_{j\ell}$, was previously defined by the materiel balance constraints. The total cost (actually a negative cost, or savings) resulting from mothballing is

$$\sum_{j=1}^{N} \sum_{\ell=1}^{Y} (-d_{j\ell}) s_{j\ell}$$

where $d_{j\ell}$ is the unit refund for mothballing a type j machine in subperiod ℓ.

Savings due to machine salvage must be calculated for both the inherited and the new equipment, as was done for operating costs. The machines represented by $x_{j\ell m}$ are salvages after $m - \ell + 1$ subperiods of service, providing that $m \neq Y$ (if $m = Y$, they would be truncated instead of salvaged). We define $e_j\,(m - \ell + 1)$ as the salvage value of a type j machine after $m - \ell + 1$ subperiods of use. The total savings from salvage is then

$$\sum_{j=1}^{N} \sum_{\ell=1}^{Y-1} \sum_{m=\ell}^{Y-1} \left(-e_j\,(m - \ell + 1)\right) x_{j\ell m}$$

Similarly the savings resulting from salvage of the inherited assets is given by the sum

$$\sum_{j=1}^{N} \sum_{\ell=K_j}^{0} \sum_{m=0}^{Y-1} \left(-e_j\,(m - \ell + 1)\right) w_{j\ell m}$$

Credit for truncation is calculated from the candidate solution at the end of the planning period; that is, when $m = Y$. Hence, the machines $x_{j\ell Y}$ are $Y - \ell + 1$ subperiods old. We define $f_{j(Y - \ell + 1)}$ as the credit added for a type j machine after $Y - \ell + 1$ subperiods. The truncation credit (introduced as a minus quantity in the cost function) is then

$$\sum_{j=1}^{N} \sum_{\ell=1}^{Y} \left(-f_j\,(Y - \ell + 1)\right) x_{j\ell Y}$$

for the purchased equipment, and

$$\sum_{j=1}^{N} \sum_{\ell=K_j}^{o} \left(-f_j (Y - \ell + 1)\right) w_{j\ell m}$$

for the inherited equipment.

This completes the development of the cost expressions; the entire cost function is shown in Fig. 3-1.

THE SOLUTION PROCEDURE

There are two special restrictions in the cost function that are essential to the branch-and-bound solution procedure. First, the function must be "separable" meaning that each variable must be priced separately (each cost expression must be a function of a single variable) -- this is the reason for the development of the master variable relation constraint; otherwise the R&D and procurement costs would be a function of a sum of variables. The second restriction is that all of the separable functions which make up the cost function must be concave*. For functions of a single variable, this means that a straight line drawn between any two points on the graph of the function always has a value which is less than or equal to the value of the function. A linear function is then a special case of a concave function.

---

*In mathematical notation a function $f(x_1, \ldots, x_n)$ is concave if for any two points $(x_1', \ldots, x_n')$ and $(x_1'', \ldots, x_n'')$, and any t, $0 < t < 1$, $f[tx_1' + (1-t)x_1'', \ldots, tx_n' + (1-t)x_n''] \geq tf(x_1', \ldots, x_n') + (1-t) f(x_1'', \ldots, x_n'')$.

Fig. 3-1 COST FUNCTION

$$\phi(x) = \sum_{j=1}^{N} U_j(x_j) + \sum_{j=1}^{N} a_j x_j^{b_j} + \sum_{j=1}^{N}\sum_{\ell=1}^{Y} (-d_{j\ell})s_{j\ell}$$

R&D costs | Procurement costs | Savings due to mothballing of unneeded vehicles

$$+ \sum_{j=1}^{N}\sum_{\ell=K_j}^{0}\sum_{m=1}^{Y}\sum_{k=2-\ell}^{m-\ell+1} c_{jk} w_{j\ell m} + \sum_{j=1}^{N}\sum_{\ell=1}^{Y}\sum_{m=\ell}^{Y}\sum_{k=1}^{m-\ell+1} c_{jk} x_{j\ell m}$$

Operating and maintenance cost

(inherited fleet) (purchased fleet)

$$+ \sum_{j=1}^{N}\sum_{\ell=K_j}^{0}\sum_{m=0}^{Y-1} (-e_j (m - \ell + 1)) w_{j\ell m} + \sum_{j=1}^{N}\sum_{\ell=1}^{Y-1}\sum_{m=\ell}^{Y-1} (-e_{j(m-\ell+1)}) x_{j\ell m}$$

Savings resulting from vehicle salvage

(inherited fleet) (purchased fleet)

$$+ \sum_{j=1}^{N}\sum_{\ell=K_j}^{0} (-f_{j(Y - \ell + 1)}) w_{j\ell Y} + \sum_{j=1}^{N}\sum_{\ell=1}^{Y} (-f_{j(Y - \ell + 1)}) x_{j\ell Y}$$

Savings due to crediting the value of fleet owned at the end of the planning period

(inherited fleet) (purchased fleet)

In addition, there are three special characteristics of the constraint set which should be noted: (1) all of the constraint equations are linear, (2) the constraint set is closed*, and (3) the problem variables can be bounded from above and below. It can be shown that the problem as described above is but a special case of the general non-convex programming problem addressed by Falk and Soland[2]. We can apply their solution algorithm to this problem by solving a sequence of linear programming (LP) sub-problems.

We define a vector of upper bounds $L = (L_1, L_2, \ldots, L_j, \ldots, L_N)$ on the master variables x_j such that

$$0 \leq x_j \leq L_j;\ j = 1, 2, \ldots, N$$

and note that these bounds can be obtained directly from the input tableau (e.g., Fig. 1-3). These lower and upper bounds define a set of N closed intervals which we will refer to as set C. We need only explicitly define upper bounds on the master variables x_j since these are the only bounds used in the solution procedure—the remaining problem variables have implicit bounds which can be derived from the master variable bounds for the x_{jlm} variables, from the consistency constraints for the p_{ikl} variables, and from the vintage constraints for the w_{jlm} variables.

Linear Envelopes

Having defined the set C, we next construct a "linear envelope" for the cost function across that set. This means that for each nonlinear contribution to the cost function, we define a linear function which is

---

*Specifically, a closed set is one in which the boundaries are included in the set; that is, there are no constraint equations involving strict inequalities.

the best linear underestimate of the concave function. This can be done graphically by constructing the straight line from the two end points of the concave function across the interval defined by C.

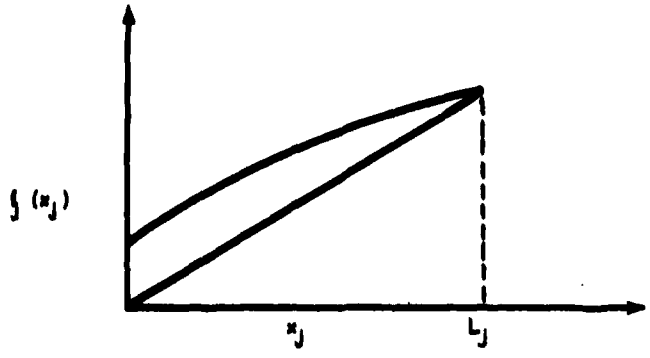


Fig. 3-2—A Linear Envelope of a Concave Function with Discontinuity at Origin

We next construct an estimating sub-problem by replacing each concave cost function with its linear envelope. Our objective function is now linear and thus a linear programming problem is defined. This linear programming problem can, of course, be solved but we will not discuss its solution in detail. We only mention that a "revised simplex" algorithm incorporating "generalized upper bounding" is used. A detailed description of this method is presented by Beale.[3]

Bounding the Solution

Since any solution to the linear problem underestimates the cost function of our original problem for that solution vector, the solution to the linear programming problem provides a lower bound on the solution of the non-linear programming (NLP) problem. This can be shown algebraically by letting f represent the objective function of the LP, then

$$f(x) \leq \phi(x)$$

RAC

for all $x \in C$. Now if we let x' represent the solution vector to the LP, and x^* represent the solution vector to the NLP, then since

$$f(x') \leq f(x)$$

and

$$\phi(x^*) \leq \phi(x)$$

for all $x \in C$, then

$$f(x') \leq f(x^*) \leq \phi(x^*) \leq \phi(x)$$

or, most importantly

$$f(x') \leq \phi(x^*)$$

This also implies that the solution to the LP in a given subset is a lower bound to the NLP within that subset.

Similarly, it can be argued that the actual value of the non-linear cost function at the solution vector of the LP is an upper bound on the solution to the NLP. That is,

$$\phi(x') \geq \phi(x^*)$$

Therefore, bounds can be established on the solution to the problem since

$$f(x') \leq \phi(x^*) \leq \phi(x')$$

The object of the algorithm is to find a solution vector x^k by solving a sequence of m sub-problems and to show that

$$\phi(x^k) \leq \phi(x^i) \; ; \; i = 1,2, \ldots k, \ldots m$$

and

$$f(x^j) \geq \phi(x^k)$$

for all j such that the subsets C_j form some partition of C. It has been shown that this sequence of sub-problems is finite (Falk and Soland, Theorem 3) and that x^i converges to, and terminates at x^* for the restrictions

RAC

previously stated.

Partitioning into Subsets

The sequence of sub-problems are generated by a branch-and-bound procedure. This procedure successively partitions the original set C into smaller and smaller subsets, each of which is defined by upper and lower bounds on the master variables. Further, each subset defines a LP sub-problem and its linear envelope provides a better underestimate across the subset than that provided by the linear envelope of the preceding problem. The specific method of partitioning is the Falk-Soland[2] "weak refining rule". Given that we have the solution to any sub-problem $x^m = (x_1^m, x_2^m, \ldots, x_n^m)$, then the weak refining rule states that one should select the partitioning variable to be x_j^m if the difference between the value of the concave function at x_i^m and the value of the linear underestimate at x_i^m is at least as great as the corresponding difference for any other x_j^m, where $j \neq i$. That is, select that x_i^m which satisfies

$$\phi_i (x_i^m) - f_i (x_i^m) = \text{Max} \left\{ \phi_j (x_j^m) - f_j (x_j^m) : \quad j = 1,2, \ldots, N \right\}$$

This gives us the variable on which to branch and partition the set C_m into two subsets C_m' and C_m''. In general the set C_m is defined by

$$\ell^m \leq x \leq L^m$$

where ℓ^m is the lower bound vector, L^m is the upper bound vector, and $\ell_i^m \leq L_i^m$ for all i. We wish to divide the interval $[\ell_i^m, L_i^m]$ for the variable x_i at the value x_i^m. This means that the variable x_i will range across the sub-interval $[\ell_i^m, x_i^m]$ in set C_m', and across the sub-interval $[x_i^m, L_i^m]$ in set C_m''. Then set C_m'' is defined by

$$\ell^{m''} = (\ell_1^m, \ell_2^m, \ldots, x_i^m, \ldots, \ell_n^m) \leq x \leq L^{m''} = L^m$$

and the set C'_m by

$$\ell^{m'} = \ell^m \leq x \leq L^{m'} = (L_1^m, L_2^m, \ldots, x_i^m, \ldots, L_n^m)$$

We have now partitioned the set C_m into two subsets and in effect created two new sub-problems to be solved.

The Algorithm

Thus far, each major procedure of the algorithm has been discussed. We now show their relationship to each other by presenting the basic cycle of the algorithm (see Fig. 3 -3). We begin by assuming that we are about to partition a sub-problem defined by the subset C_m, and that we have determined (by the weak refining rule) the variable x_i^m on which to branch. Two new subsets C'_m and C''_m are formed. We then construct the linear envelope across C'_m (block 5 of Fig. 3-3) and solve the corresponding LP to obtain the solution vector $x^{m'}$ and the cost $f(x^{m'})$. We determine a variable $x_i^{m'}$ on which to branch. We next compute the value of $\phi(x^{m'})$ and compare this value with the value of $\phi(x^o)$, where $\phi(x^o)$ is the best solution found thus far (the reference solution). If $\phi(x^{m'}) < \phi(x^o)$, then we update and replace $\phi(x^o)$ with $\phi(x^{m'})$. We then store the problem description along with the values $x_i^{m'}$ and $f(x^{m'})$ on a branching list. The process is repeated for the subset C''_m and its information is stored on the branching list. We then select and remove that sub-problem, C_α, from the branching list such that $f(x^\alpha) = \text{Min}\left\{f(x^k); \text{ for all } k\right\}$ and the cycle repeats.

Given just this cycle, the algorithm would run forever. We need, therefore, define some type of termination criteria. First, if when we solve the LP problem defined by C_m, we find that $f(x^m) \geq \phi(x^o)$ then we

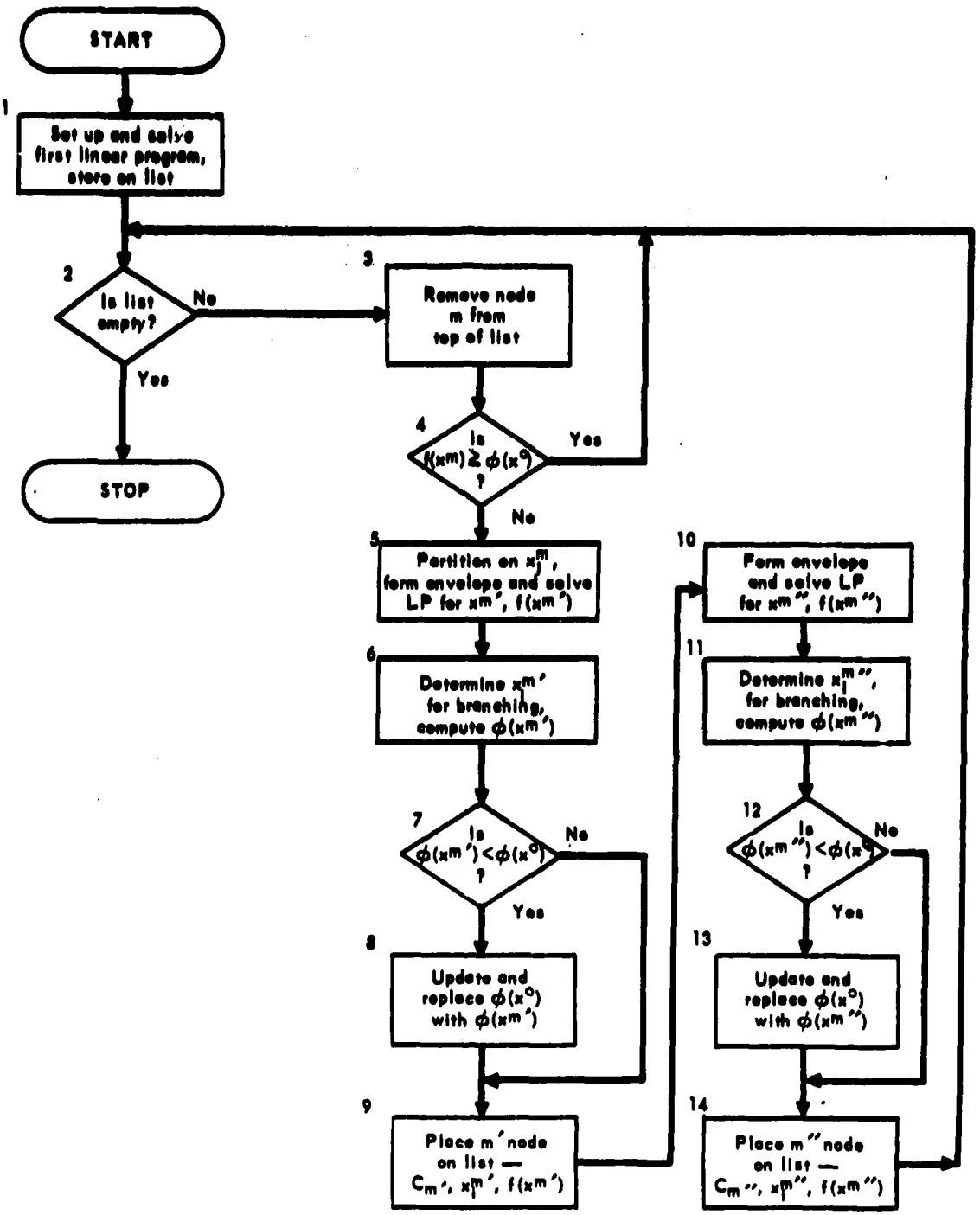


Fig. 3-3—The Algorithm

know that no solution better than $\phi(x^o)$ exists in the subset C_m. So we discard the subset from further consideration (block 4 of Fig. 3-3) and select the next problem on the branching list. Finally, when there are no more problems on the branching list, we know that

$$\phi(x^o) = \phi(x^*), \text{ and } x^o = x^*, \text{ since } \phi(x^o) \leq \phi(x) \text{ for all } x \text{ in } C.$$

In Theorem 3 of their paper[2], Falk and Soland prove that this algorithm converges to the solution of the problem with a finite number of sub-problems. It should be noted that the term finite does not concern itself with the number of sub-problems required for convergence, and that in some cases it may be expedient to only ensure that the solution obtained is within some small tolerance of the optimal solution (for example: within 0.1% of the optimal solution). This is easily accomplished by adjusting the termination criteria to discard a subset if

$$f(x^m) \geq \phi(x^o)/(1 + \text{tolerance factor})$$

It is considered important to review and to emphasize here that implementations of this optimization model, and the resulting solutions, are subject to certain assumptions which must be carefully observed and understood. These assumptions are necessary for either of two reasons: (1) the alternatives to the particular assumption are economically prohibitive, or (2) the technical restrictions in the algorithm itself cannot be waived by any reasonable scheme. A brief review of the model assumptions, and a reference to their detailed descriptions follows:

(1) The cost-quantity relationship of a particular type vehicle (machine) must be concave and independent of the state of a vehicle of another type — see pages 3-5 and 3-10.

(2) Vehicles used to satisfy the requirements of one mission group cannot be used (pooled) in any other mission group --- see page 1-2.

(3) All convex combinations of the finite list of user-supplied equally effective alternatives are also equally effective --- see page 3-4.

Chapter 4

AN OPERATIONAL DESCRIPTION OF THE BRANCH AND BOUND SOLUTION PROCEDURE

Shortly after the computer program development began it became apparent that because of the complexity of the solution procedure and the magnitude of the problems to be solved, it would be necessary to sub-divide the solution procedure into two separate programs; namely, the matrix generator (GENLCP for GENerate Least Cost Phasing) and the main program (BBCAV2 for Branch-and-Bound conCAV functions version 2). However, as work on BBCAV2 progressed, it was seen that the output was very much technically oriented and that this orientation was essential for debugging, detecting input errors, and tracing the flow through the development of the "branching tree." Because of the necessity of this output and the desirability for a less technical report, a third program (REPGEN for REPort GENerator) was developed to display the optimal and relevant off-optimal solutions in a manner which would be easy to understand. Thus the present operational system consists of these three programs named with the mnemonics GENLCP, BBCAV2, and REPGEN.

PROGRAM INTERFACING

The major problem encountered by using separate programs is that of passing large data files from one program to another. The matrix generator creates the matrix file which must be passed to the main program and a reference list file which must be passed to the report generator so that it can process the solutions. The main program then creates a solution file which of course must be passed to the report generator. The

exact content and structure of these files will be discussed in a later section of this chapter.

The interfacing of the programs is solely dependent on the transfer of these files, and this is accomplished by the use of either a physical or a logical tape file which is manipulated by means of the operating system control cards.*

REPGEN is easily executed within the same computer job as EBCAV2, therefore there are at least two job configurations available to the user. They are:

(1) GENLCP can be executed first as one job, while programs EBCAV2 and REPGEN are executed later as two programs within a second job. See Figure 4-1a for the CDC 6400 control cards.

(2) GENLCP, EBCAV2, and REPGEN can be executed as three different programs within one job. See Figure 4-1b for the necessary CDC6400 control cards.

Each job has the following control card types; a job, a RUN(S), and a LGO card. The job card merely describes the job to the system, the RUN(S) card executes the FORTRAN compiler, and the LGO card loads and executes the compiled program. Each job also has a request card, which requests the same tape, meaning that the x's in the parentheses are replaced by the same tape number on all request cards.

Upon execution the matrix generator creates the matrix on file 9 (TAPE9), and the reference list on file 7 (TAPE7). After the physical

---

*This report describes the use of CDC 6400 control cards for tape file manipulation. Implementation of this program(s) on other computing machines will require appropriate changes to the control card set.

MATRIX GENERATOR, MAIN PROGRAM AND REPORT GENERATOR

```
"Job Card"
RUN(S)
LGØ.
REQUEST, TAPEA, HI.  (XXXX/RING)*
REWIND (TAPE7, TAPE9, TAPEA, LGØ)
CØPYCF (TAPE9, TAPEA)
ØOPYCF (TAPE7, TAPEA)
REWIND (TAPEA)
RUN(S)
LGØ.
REWIND (TAPE8, TAPEA, LGØ)
CØPYCF (TAPEA, DM1, 2)
CØPYCF (TAPE8, TAPEA)
REWIND (TAPEA)
CØPYCF (TAPEA, DM1)
RUN(S)
LGØ.
```

---

* If TAPEA is only a logical tape file that is not saved at the end of this job, then this control card can be omitted.

Fig. 4-1a Controls Cards to Execute All Programs as a Single Job

MATRIX GENERATOR

```
"Job Card #1"
RUN(S)
LGØ.
REQUEST, TAPEA, HI.  (XXXX/RING)
REWIND (TAPE7, TAPE9, TAPEA)
CØPYCF (TAPE9, TAPEA)
CØPYCF (TAPE7, TAPEA)
```

MAIN PROGRAM AND REPORT GENERATOR

```
"Job Card #2"
RUN(S)
REQUEST, TAPEA, HI.  (XXXX/RING)
LGØ.
REWIND (TAPE8, TAPEA, LGØ)
CØPYCF (TAPEA, DM1, 2)
CØPYCF (TAPE8, TAPEA)
REWIND (TAPEA)
CØPYCF (TAPEA, DM1)
RUN(S)
LGØ.
```

Fig. 4-1b Control Cards to Execute Programs as Separate Jobs

tape is requested, the tape and both files are rewound to their load points by the REWIND card, and then the files are copied, matrix file first, onto file A (TAPEA) in coded, rather than binary, format through the use of the COPYCF cards.

TAPEA is then transfered to the main program and the first file is read during execution. The main program creates the solution file on TAPE8. Upon completion of the execution of RBCAV2, TAPE8 and TAPEA are rewound. The matrix and reference list files on TAPEA are then copied onto dummy files in order to space to the end of the tape file and the solution file is copied onto the end of TAPEA.

The report generator only uses the last two files on the tape, so before it executes the first file it is copied onto a dummy file. The program then reads the last two files as it executes. The total interfacing of the programs is shown in the macro flowchart of the system, Fig. 4-2.

THE MATRIX GENERATOR

Program Logic

This program produces as its main output the matrix of coefficients for the phasing problem. Essentially the program reads the input deck and then determines the number of constraints (rows) and the number of variables (columns) for the problem. In order to facilitate communication between programs and between the program and the analyst, this program assigns symbolic names to the rows and columns. These symbolic names can then be decoded to determine exactly what that column (row) represents. The naming convention which the program uses is shown in Fig. 4-3. These symbolic names are generated through the use of the array NP. This array contains a positive integer code for the integers 1-288 (288 is the largest integer used by the program), and this code is shown in Fig. 4-4. This

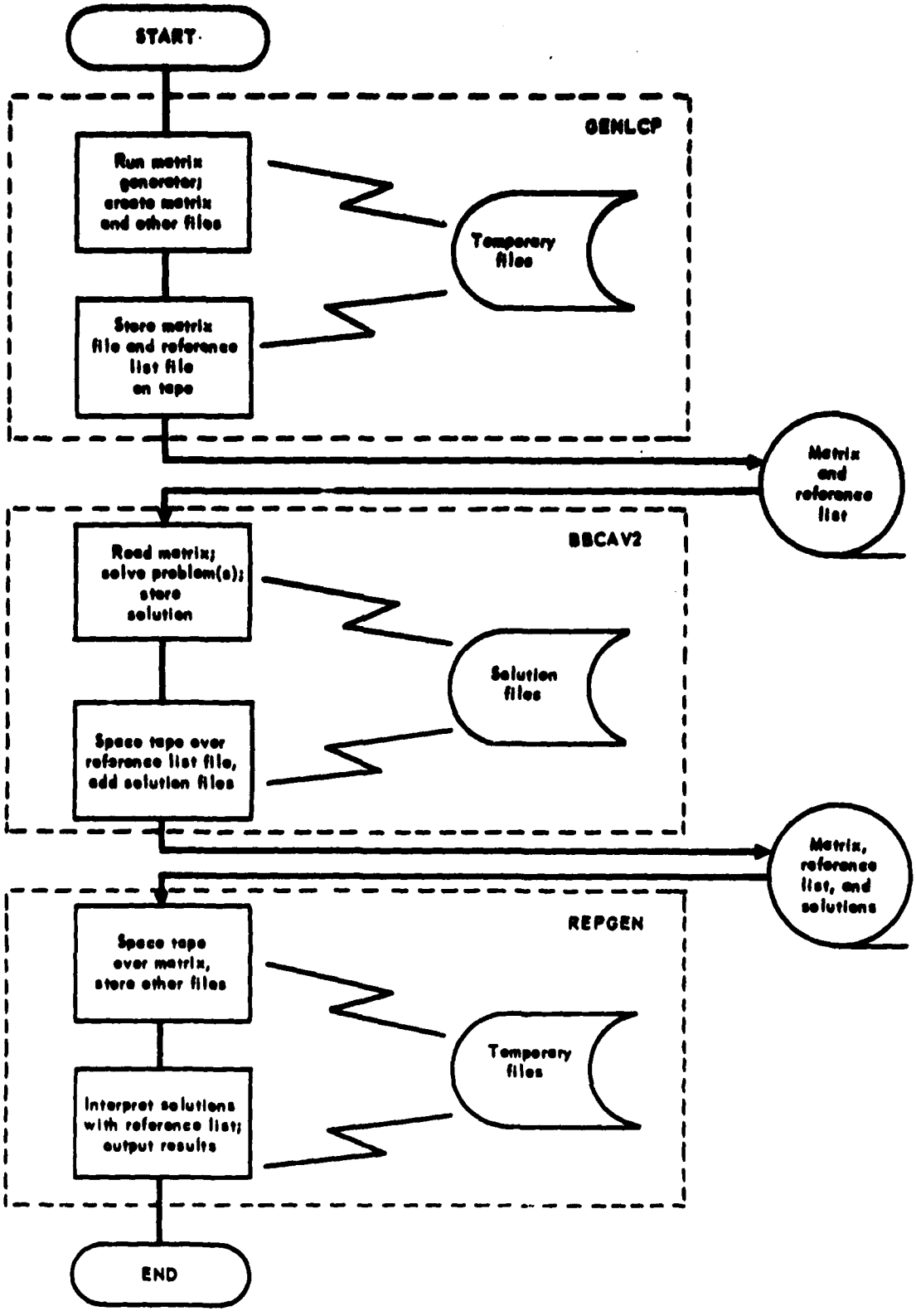


Fig. 4-2—System Macro Flowchart

Row Type	Symbol
Master Variable Relation	$SUMX_j$
Procurement Constraint	PC_ℓ
Vintage Constraint	IW_jP_ℓ
Materiel Balance Constraint	X_jP_ℓ
Consistency Constraint	T_iP_ℓ
Objective Function	COST

Column Type	Symbol
Master Variable	X_j
Excess Procurement Funds	P_ℓ
Inherited Fleet Variable	$W_{j\ell'm}$
Alternative Selection (or Proportion)	$P_{ik\ell}$
Purchased Fleet Variable	$X_{j\ell'm}$
Mothballing Variable	$S_{j\ell}$
Right Hand Side Vector	RHSI

Coding is accomplished by replacing the subscript with a one character number code where the subscript represent the following:

j = vehicle ID number

i = task ID number

k = alternative number

ℓ = period number

ℓ' = period purchased number

m = period sold number

Fig. 4-3 Symbolic Naming Convention

Integers	Codes
01 - 99	01 - 99
100 - 109	A0 - A9
110 - 119	B0 - B9
120 - 129	C0 - C9
130 - 139	D0 - D9
140 - 149	E0 - E9
150 - 159	F0 - F9
160 - 169	G0 - G9
170 - 179	H0 - H9
180 - 189	J0 - J9
190 - 199	K0 - K9
200 - 209	L0 - L9
210 - 219	M0 - M9
220 - 229	N0 - N9
230 - 239	P0 - P9
240 - 249	Q0 - Q9
250 - 259	R0 - R9
260 - 269	T0 - T9
270 - 279	U0 - U9
280 - 288	W0 - W8

Fig.4-4 Positive Integer Code

code replaces the integer values indicated by the subscript in the naming convention.

The program next determines the value of all of the non-zero elements of the matrix. As it determines these values it creates a "packed and labeled" matrix file; packed meaning it contains no zero valued elements, and labeled meaning it has associated with each element its row and column names. In order that this packed file be created in some standard format, it was decided that MPS360 format be used. This has the advantage that should one wish to use this file on other mathematical programming systems, it is in a format acceptable to both IBM and CDC systems.

After this file is created, the program then fills in the zero-valued elements and creates an unpacked and unlabeled file which is the input matrix to BBCAV2. At this point it also creates the reference list which indicates the symbolic name to be associated with each column of the unlabeled file.

Core Allocations

The program requires 107,300 octal (36,544 decimal) words of core on the CDC 6400 system in order to load. Almost two-thirds of this is used for common storage; most of which is in the form of subscripted arrays. These arrays are presently dimensioned to handle the following size problems:

(1) number of machines (vehicles) ≤ 7

(2) number of task tables ≤ 8

(3) number of periods ≤ 10

(4) number of alternatives in a task ≤ 288

(5) number of years from which vehicles are inherited ≤ 16

RAC

(6) maximum vehicle life ≤ 25

(7) maximum length of a period ≤ 6

Any of these which are exceeded will necessarily cause changes in the subscripted arrays and core storage requirements.

The core storage allocation can be broken up into the following subgroups:

		OCTAL	DECIMAL
CDC system routines	=	13,210	5,768
COMMON storage	=	51,562	21,362
GENLCP	=	20,323	8,403
MATFILL	=	1,415	781
YINTERP	=	207	135
YRCOST	=	137	95
TOTAL		107,300	36,544

User's Subroutine - YRCOST

This subroutine is used to calculate operating, mothballing, salvage and truncation costs based on the age of a vehicle. For everything but mothballing, this is accomplished by filling the array COSTS with the yearly costs for a vehicle up to an age of 30 years. The main routine just selects the appropriate cost from this array as it is needed. For mothballing, however, the routine is called each time a value is needed, and only the mothballing cost is calculated. This is done through the use of the ENTRY MOTH statement.

Operating costs are assumed to be increasing at R x 100 percent each year (not compounded), with R presently set equal to zero. It is noted that the present formulation for operating costs does not account

for the attrition of aircraft over the years; such account is handled explicitly through the materiel balance constraints in Chapter 3. If the user has 10 year operating costs which already include costs for attrition, then $V_{\ell-\ell'j}$ should be set equal to 1.0. Note also that it is assumed that no period is longer than six years. If a period is, it will be necessary to change this card in subroutine YRCØST.

IB = VLIFE (J) + 10

to

I.. = VLIFE (J) + (N - 1)

where N is the number of years in the longest period (IB cannot exceed 30 in any case). Salvage values are calculated to be alpha to the ith power times the initial salvage value, where i is the last year of service, with alpha currently set to 0.5. Truncation value is assumed to decrease linearly from the initial salvage value to zero over the expected lifetime of the aircraft. Mothballing savings are assumed to be R1 x 100 percent per year of the first year operating cost. This account of mothballing savings will be in error (low) for aircraft mothballed beyond the first year of their life; however, this error will not be serious unless the increase in operating cost per year is large. An alternative and exact method of accounting for savings due to mothballing was considered but the added complexity due to its implementation was not considered to be worthwhile. The present value of R1 is 0.90. Under the present system, actual costs are used instead of costs discounted to present value.

Input Formats

The input consists basically of three types of data sets: (1) vehicle tables--describe availability, vintage, life, and cost information

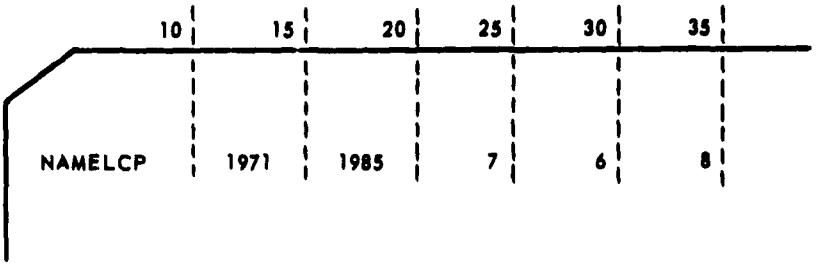
for each vehicle type; (2) period tables--describe length, money available, and tasks for each period; and (3) task tables--describe vehicles which can be used for each task and alternative ways of accomplishing the task with those vehicles. In addition to these tables there is a title card and a group of header cards which inform the program of the type of tables that follows. The formats for all of the cards are given in Figs. 4-5 through 4-8 and reference to these will be helpful for the discussion which follows.

The title card sets the major parameters for the problem. The first field contains the name of the run being made. This name is used as the identification on the tapes produced by the program. This field is stored as alphanumeric characters and should be left-justified on the card to avoid leading blanks. The remaining fields on the card are all right-justified in integer format and contain, in this order, the first year of the planning horizon, the last year of that horizon, the number of vehicle tables for the problem, the number of task tables, and the number of period tables.

The header cards have only one alphanumeric field. In this field is placed one of four words, either the name of the type table which follows that card (vehicle, period, or task) or the word ENDTABLE which marks the end of the data deck. It is essential that each table have a header card and that there are exactly as many tables of each type as was called for in the title card.

The vehicle tables have three basic card types. The first, the name card, has three fields; an alphanumeric field for the vehicle name, and two integer fields, one for the first year that the vehicle can (or did) enter the fleet and the other for the maximum life of that vehicle

TITLE CARD



HEADER CARDS

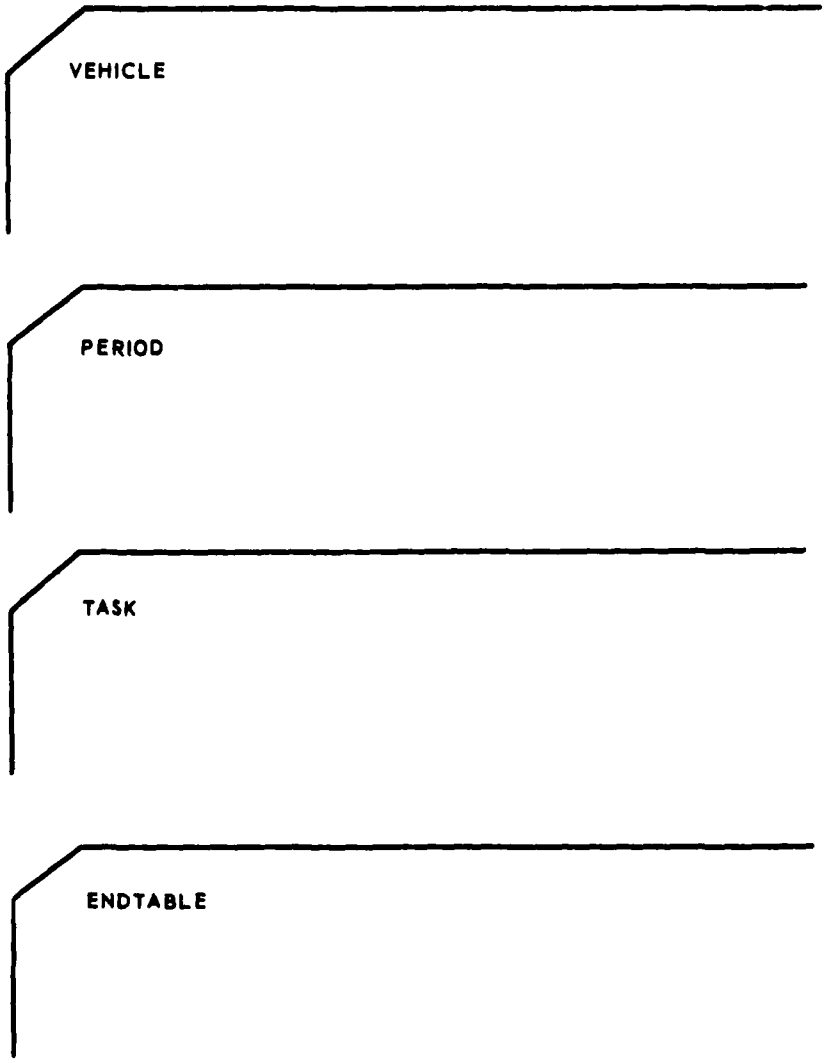
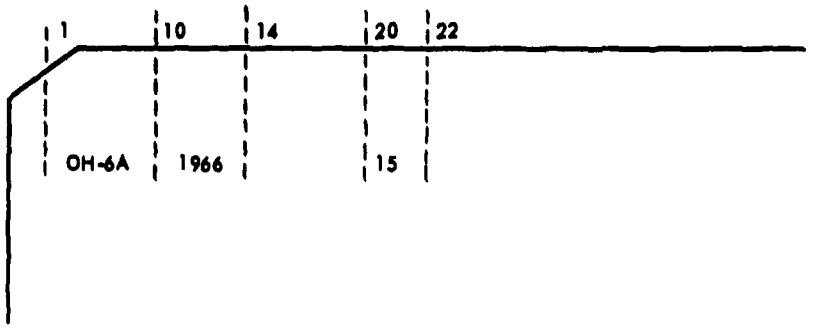
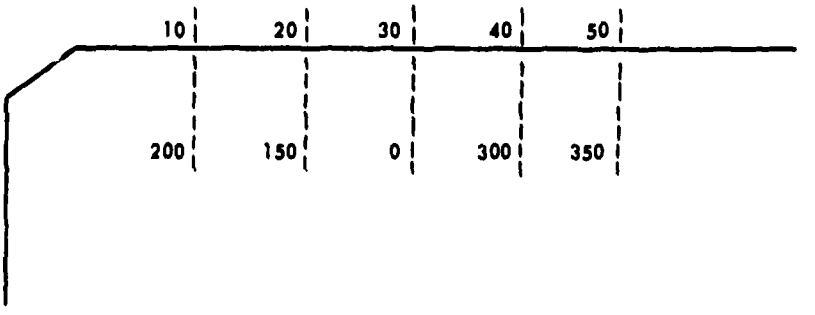


Fig. 4-5—Basic Cards

CARD 1



CARD TYPE 2 (only for inherited vehicles)



CARD TYPE 3

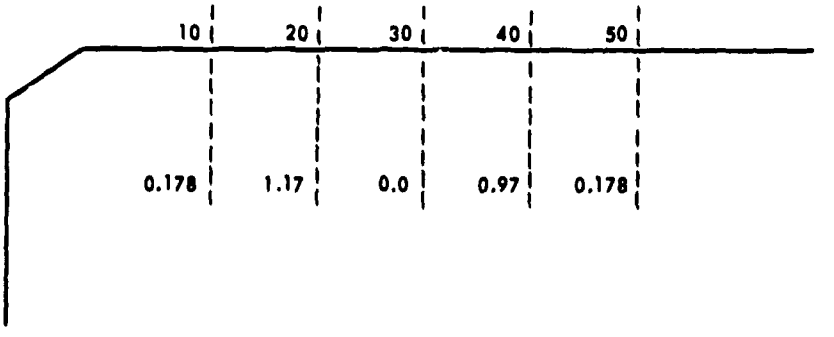
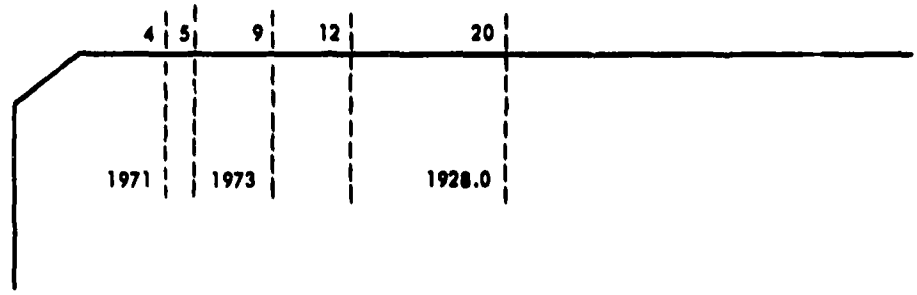
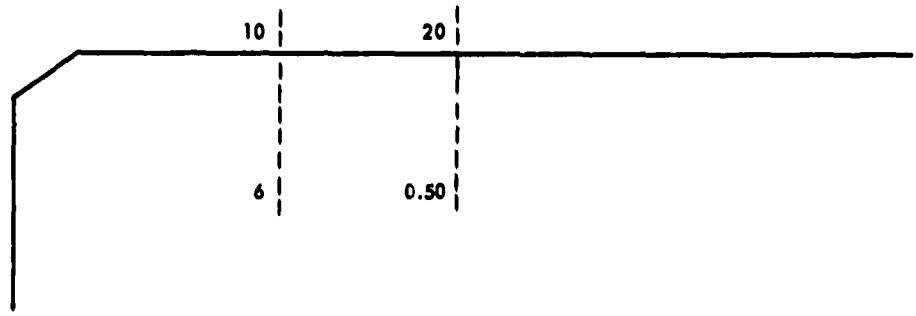


Fig. 4-6—Vehicle Table

CARD 1



CARD 2



CARD TYPE 3

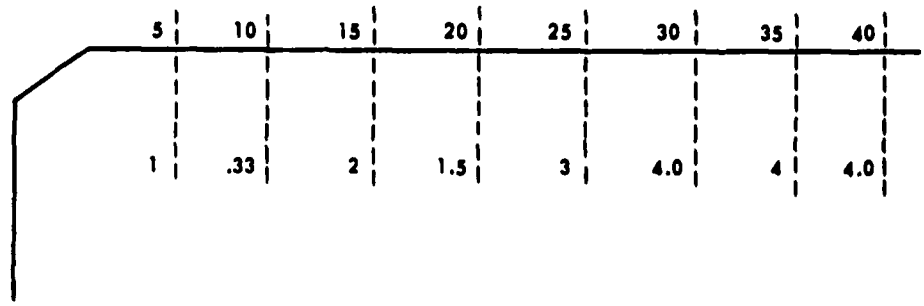
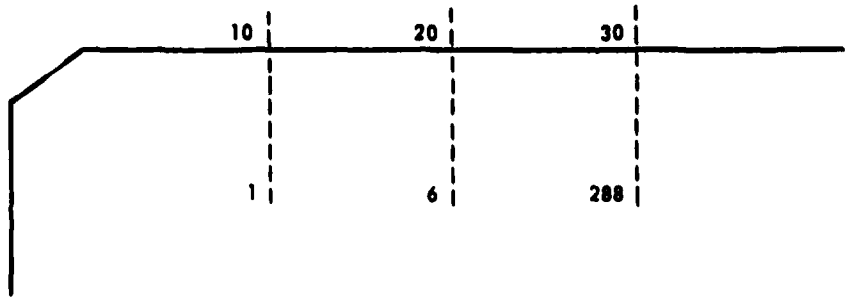


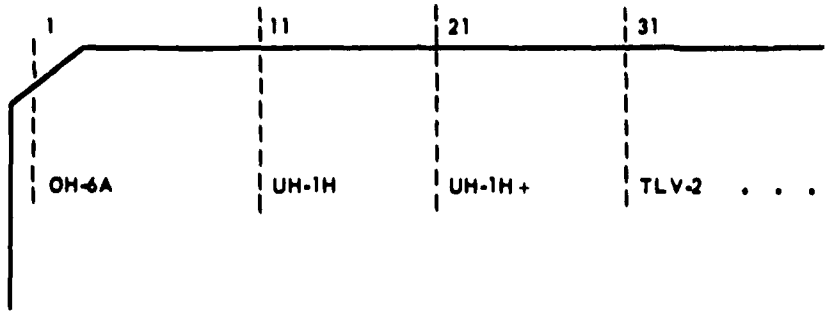
Fig. 4-7—Period Table

RAC

CARD 1



CARD 2



CARD TYPE 3

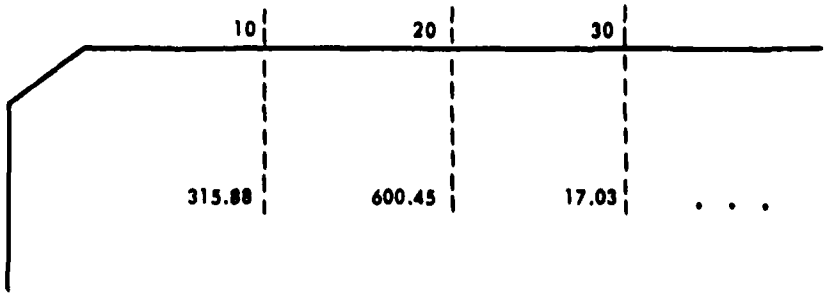


Fig. 4-8—Task Table

type. The second type of card is only for those vehicles which are presently in the fleet, and they are used to describe the number and age of the vehicles inherited by the problem. Each card has eight ten-digit integer fields, and each field contains the number of vehicles of the type described on the first card which were purchased in some specific year. The first field on the first card of the second type contains the number inherited from the first year that the vehicle entered the fleet (that year which is listed in field two of card type one). Each field represents the years sequentially from that year to the last year prior to the starting year of the planning period, so that if a vehicle entered the fleet 10 years prior to the planning period, it would require two cards to describe that vehicle's contribution to the fleet; the first would have all eight fields filled and the second would have the first two fields filled. The last card in a vehicle table is the cost card, which contains five ten-character floating point fields. The first is a cost (the initial salvage value) to be used with the salvage and truncation equations; the second is the ten year operating cost; the third is the research and development cost; the fourth is the retention rate (1- the attrition rate); and the last field is the unit cost associated with the linear estimate of the procurement cost. In general, the first and last fields will contain the same number, although this is not essential. All of the costs in each vehicle table should be scaled by the same factor, such as 10^6, so that the ten year operating costs fall roughly between 1 and 10 for all of the vehicles. This will give the best results in BBCAV2 by avoiding numerical roundoff errors.

The period tables are also input with three card types, except for inherited (past) periods which use only the first card type. The first

card contains three fields; the first two are integer fields which contain the first and last year of the period respectively (if a period is only one year, these two entries would be the same), and the third field is a floating point field containing the procurement cost constraint level for that period. If the procurement constraints are not desired, then an extremely high number should be placed in the third field of the first card for the first period, so as to keep the constraints from being binding. Of course, no constraint need be given for the inherited periods (the periods preceding the start of the planning period). The second card contains a ten-character integer field for the number of tasks to be done in that period and a ten-character floating point field for a scale factor (in that order). The scale factor is applied to all of the tasks and can be thought of as a representation to the growth in mission requirements from one period to the next. The third card type has 16 five-character fields which are used in pairs. The first field of each pair is an integer task identification number, and the second field is an individual scaling factor related solely to that task. This factor might be used for the number of times the task must be independently performed in that period or for other similar multiplicative factors. If more than eight tasks are to be performed in a period then a second card type three must be used.

The final type of table is the task table. Its first card has three ten-character integer fields which contain the task identification number, the number of columns in the table, and the number of rows (alternatives) in that table. The second card contains eight ten-character alphanumeric fields for the vehicle names associated with each column. These names are left-justified and are the same names as appear in a

vehicle table. Card type three is used for inputing the alternative ways of performing each task, and each card represents a different alternative, so that there are exactly as many cards as there are alternatives given in field three of card one. Card type three has eight ten-character floating point fields. Each field contains the number of the type vehicle represented by that column which is used in accomplishing the task under the alternative given by that card.

There are only two limitations on how the data deck can be organized. First, a task table must not precede the vehicle table for any vehicle referenced in that task table, and second, although the period tables may be separated by other tables, they must be entered in chronological order. However, to avoid any problems, the structure shown in Fig. 4-9 is recommended.

Output Description

The final output of the program includes a tape which contains two files; the unlabeled matrix file and the reference list file. The matrix file is composed of four parts; a record containing the number of rows and columns in the matrix and the upper bounds for the non-linear variables, a title record, a row descriptor record, and the matrix. The initial record is formatted (2I6/(6F12.4)). The title card is formatted by (A4, 10X, A8), where the first four alphanumeric characters contain the word "NAME" and the last eight have the problem title. The row descriptor is a vector which tells the main program whether the row is the type equal (0), less than or equal (1), greater than or equal (2), free (3), or generalized upper bound (4). It is formatted as (I12). The elements of the matrix follow this vector and have the format (F12.4), and are blocked into records by column. The reference list file has no

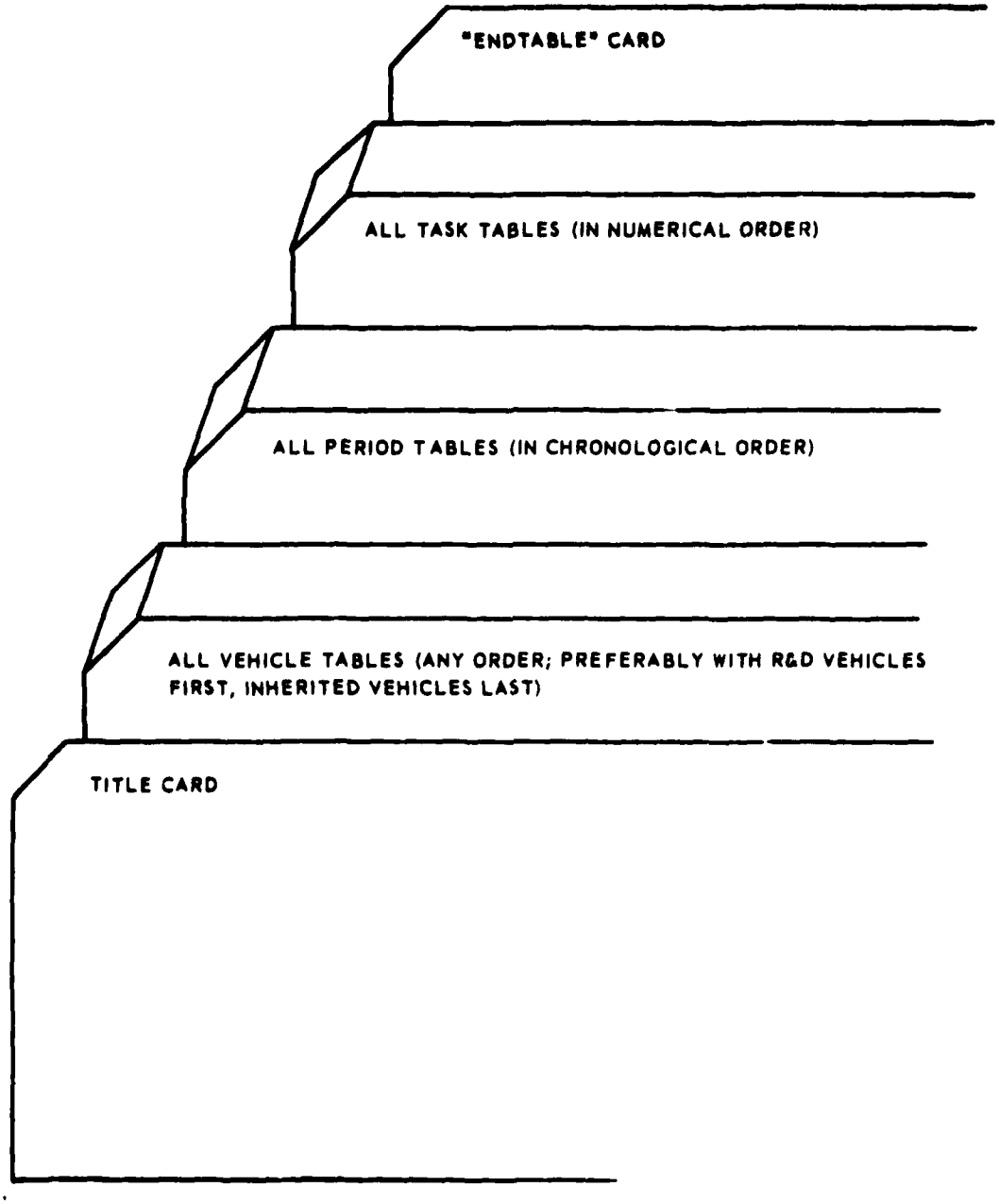


Fig. 4-9—Deck Structure for Matrix Generator

RAC

header cards and is formatted totally by (I5, 4X, A7), where the first field contains the column number and the last field the column name.

The printed output is rather lengthy, being composed of four major sections. The first section is merely a tracer through the input deck, so that if an error is found in the formats, the location of that error is fairly well known. The second section is a partial listing of the MPS360 file which can be used for spot checking the values in the file. The third section is the cross reference list for the column numbers and names. At the head of this list is the row descriptor vector and at the end is a small section of data (number of rows and columns, and the upper bounds on the vehicles) which are needed for input to BBCAV2. The final section is a documentation listing of all of the input data for the run.

THE MAIN PROGRAM

Program Logic

The method of solution is of course an implementation of the algorithm presented in Chapter 3. The first node, which represents the linear underestimate of the entire problem, is solved as a special case of the general solution procedure by the routine BOX1. This routine establishes a branching tree (containing only one node), sets values for certain key parameters, and defines several arrays necessary for the algorithm. All subsequent nodes are solved in pairs by examining both branches from an existing node on the tree.

Although the algorithm guarantees that an optimal solution can be found after the solution of a finite number of nodes, the program is not usually allowed to run until this has been proven! There are several reasons for this--the main one being that the time, and hence money, involved

in finding this solution may be extremely large. Also of importance when dealing with large problems having many near-optimal solutions is the problem that it may be impossible to determine which solution is indeed optimal because of roundoff and other small numerical errors. Because of this the program is written so that it will solve for a solution which is only guaranteed to be within some tolerance of the optimal solution. This tolerance level is an input parameter (ϵ) which is used to insure that optimal solution times $1 + \epsilon$ is greater than the current best solution before the algorithm terminates. It is recommended that $\epsilon \geq .00001$ be used to avoid the problems of numerical errors.

When solving any given subproblem, the program performs a simple transformation on the subproblem before solving it by means of the LP. This transformation involves shifting the axes of the problem such that all of the lower bounds for the LP problem are zero. The sum of the costs of the non-zero lower bounds are stored as the variable EKO. The actual lower bound to the node is then the cost of the LP problem plus the value EKO, and the actual values of the x-vector are given by the LP x-vector plus the lower bound vector. This simple transformation is used to simplify the LP solution procedure by eliminating the need to check lower bounds.

The program makes use of the "ordered list" concept for selecting a node for branching. However, the list itself is not literally ordered so that each time a node is sought the list is searched to determine which node has the least lower bound. This node is then removed from the list leaving a gap which is marked but not filled. Then when new nodes are created, any gaps are filled before making the list longer. The procedure terminates when the least lower bound on the list is greater than the best solution found divided by $1 + \epsilon$.

Because of its size, the matrix of coefficients is stored on a disk file, however certain parts are kept in core at all times. The b-vector [B0(100)] is kept in core so that the changes in the right-hand-side based on changing the lower bounds of the master variables can be rapidly made for each node. The cost vector [C(10)] for the non-linear terms is also stored in core since these elements are changed at each node. Finally the column data for each of the non-linear variables [T(100,10)] are kept in core since they are used to update the B vector.

The program has been constructed to be repetitive in order to solve for off-optimal solutions, i.e., interesting solutions near the least-cost-solution. This means that any number of problems which use the same matrix of coefficients can be run at one time. For example one could run the optimal solution and, after that is found, run the problem for the best solution for which no resources requiring R&D are allowed. The technique for doing this is discussed in the section on input formats.

Core Allocation

This program stores the main array of information, the matrix of coefficients, on high-speed auxiliary storage (disc). However in addition to this, the program has almost 45,000 octal words of common core storage. The dimensions of the arrays in these common areas are dependent on only three parameters:

(1) number of rows in matrix $\leq$ 100

(2) number of nonlinear variables $\leq$ 10

(3) number of nodes on branching list $\leq$ 25

The total core storage requirement is 114,251 octal words on the CDC 6400 system, which is subdivided as follows:

	OCTAL	DECIMAL
CDC system routines	7,312	3,786
COMMON Storage	44,514	18,764
BBCAV2	22,063	9,266
BOX 1	334	220
GETASQ	140	96
GETC	202	130
GETPHI	320	208
INITA	375	253
NXBRN	303	195
PARAMS	241	161
PRESET	131	89
READIN	65	53
SET	15	13
TABOUT	233	155
TIMEC	126	86
LP	430	280
BOUND	60	48
COLUMN	560	368
DISC	412	266
DOT	70	56
ESCAPE	362	242
EXITS	72	58
FEASCH	522	338
INVERT	477	319
IO	666	438
KEYCH	232	154
KEYFND	124	84
MAPIN	564	372
MAPOUT	1,135	605
PIVOT	223	147
PRIMAL	620	400
ROW	467	311
SETEND	73	59
SETUP	361	241
STATUS	465	309
XCHECK	774	508
	114,251	39,081

USER'S SUBROUTINE

GETPHI

It is through this subroutine that the user inputs his non-linear cost functions. The routine uses these functions to determine the cost of specific numbers of each resource, or to determine the total cost of a

solution for a specific node. These functions are defined in the section of the routine between statement 150 and statement 140.

The functional value (cost) is denoted by PHI(I) for the Ith resource. It is described as a function of XPHI(I), where XPHI(I) is the independent variable representing the number of Ith type resources purchased. The examples shown in the listing are of the form:

$$\phi_i = R_i + \alpha_i x_i^{\beta_i}$$

where R_i is the R&D cost, α_i is the linear cost coefficient, and β_i describes the "learning" rate on cost as purchase size increases.

This routine should be put together using the following guidelines:

(1) The (I.GT.7) phrase in statement 150 should have the 7 changed to be the number of non-linear functions (NCF) to be described in the routine.

(2) This should be followed immediately by a computed go to statement of the form

```
GO TO (101, 102, ..., 100 + NCF), I
```

(3) If the R&D cost for the Ith resource is zero, then the section describing its cost function should contain two cards of the form

```
n  PHI(I) = ...
   GO TO 200
```

where n is the statement number such that n = 100 + I.

(4) If the R&D cost is greater than zero, then an additional card is needed so the form of the section is

```
n  IF(XPHI(I).LT...0001) GO TO 140
   PHI(I) = ...
   GO TO 200
```

where n has the same meaning as above, and the IF statement prevents the inclusion of R&D cost when the system is not needed.

RAC

The program will assume that the resources are numbered 1 thru NCF according to the order printed in the output from the matrix generator, so one should be careful to associate the appropriate function with the variable for which it was intended.

Input Formats

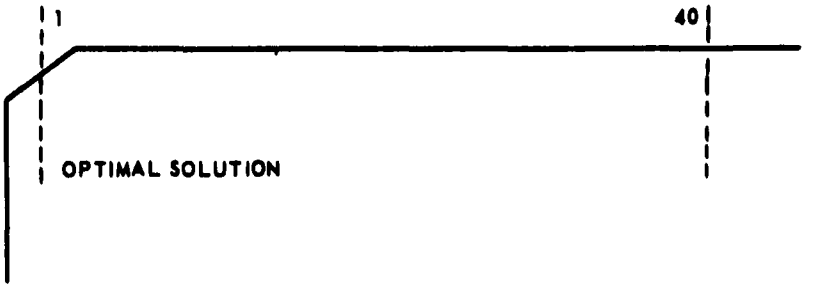
The input deck for this program is much smaller than that for the matrix generator. There are only four basic types of cards used for this program. They are shown in Figure 4-10.

The first card in the deck is the solution name card. It contains a 40-character alphanumeric phrase which describes the solution which will be achieved with this deck. This phrase is placed at the beginning of the solution file and will appear on the top of each page of output from the report generator.

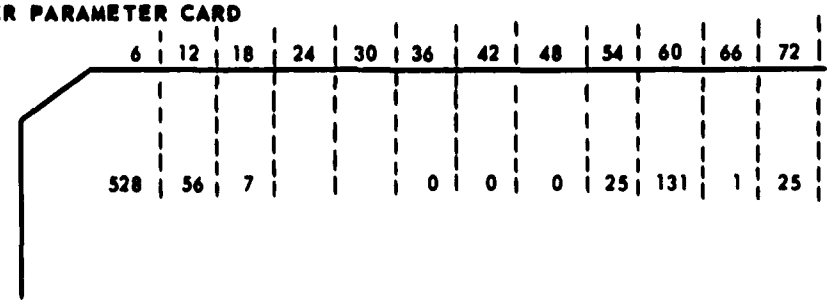
The second card is the integer parameter card. It is formatted as 12 fields, each containing 6 integer characters. The first two fields are no longer used by this program. The third field contains the number of variables (columns in the matrix) for which cost functions have been developed and programmed in GETPHI. These will be the first columns in the matrix. The ninth and tenth fields contain the preset dimensions of the array BLIST, which are currently set at 25 and 131, respectively. The fourth and fifth fields are no longer used by the program.

Fields 6, 7, 8 and 11 are all used to control the output from the program. If these fields are set to zero, the output they control will be suppressed, and if they are set to one, the output will be printed. Field 6 prints the subroutine names as they are called; it is normally used for tracing and debugging and hence should normally be set to zero. Field 7 prints the primal iterations of the linear program, and field 8

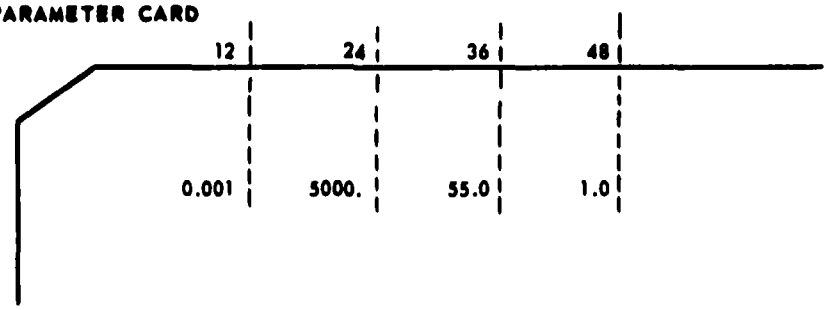
SOLUTION NAME CARD



INTEGER PARAMETER CARD



REAL PARAMETER CARD



BASIS CARDS

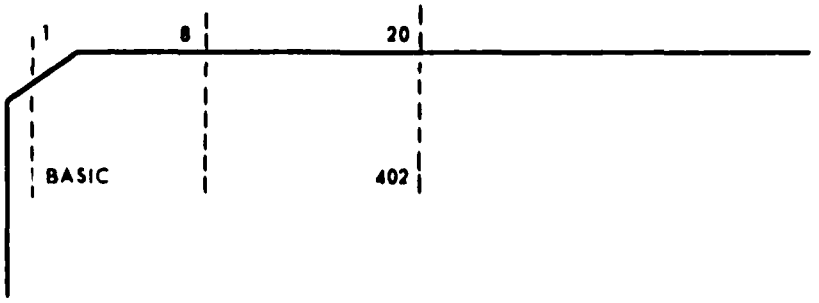


Fig. 4-10—Input Data Formats

RAC

prints the entire solution of the LP, however, field 8 is only effective if field 7 is set to one also. These two fields would normally be set to zero unless one were interested in closely examining the properties of the linear underestimates. Field 11 is the only output control normally used during production running, and it lists the column numbers and values in the solution at each node.

The last field is a termination criterion established by the user. Specifically, it is the maximum number of nodes which the program will evaluate before it outputs a solution. This can be used, for example, when one is evaluating off-optimal solutions and wishes only to get the solution corresponding to the first linear underestimate of the problem.

The third card of the input deck is the real parameter card and it has 4 fields of 12 character floating point numbers. The first field is the epsilon parameter which was previously discussed. The second field is another termination criteria; the maximum number of seconds which each solution will execute before terminating. This is not the same as the time limit on the job card since this latter is the limit on the sum of all solutions combined. The third field is the number of basis cards in the input deck (if no basis is input, leave this field blank). The last field is another output control. If this field is equal to zero, it will suppress all output except the initial problem description and the final solution (if detailed output was called for by previous options, put 1.0 in this field).

The final group of cards are the basis cards; these cards are optional and need not be included at all. However, they can be used to accomplish two important functions. First, they can be used to define an initial basic solution (which may or may not be feasible), and if this

basic solution is carefully chosen, it can greatly reduce the number of primal iterations required to find an optional solution. This is accomplished by creating one basis card for each column that one desires to have in this basic solution. The basis card is composed of an 8 character alphanumeric field and a 12 character integer field. For the basic solution, one places the word "BASIC", left-justified, in the first field and the column number in the second field. However, if the basic variable is basic in a generalized upper bound row, then the word "KEY" should be entered in the first field instead of "BASIC". If there is more than one basic variable in a GUB row, only one should be "KEY" and the rest "BASIC", since there should be exactly the same number of "KEY" variables as there are GUB rows in the matrix. The second function performed by the basis cards is the elimination of columns from the matrix. This is done by placing the word "NULL" in the first field and the column number in the second. This is useful in running off-optimal solutions, since for example, one can eliminate a resource from the mix by "nulling" the master variable, or delay the development of a resource by one period by nulling the appropriate set of purchased fleet variables.

The input deck may be constructed to produce as many solutions as desired. For each solution one creates a deck consisting of the four card types just described. These separate decks are then put together in any order to form a single input deck. The overall structure of this deck is shown in Figure 4-11.

Output Description

The output from the program can be divided into three basic groups. The first is the information related to the branch and bound algorithm, and the second group is the information related to the linear program. Both

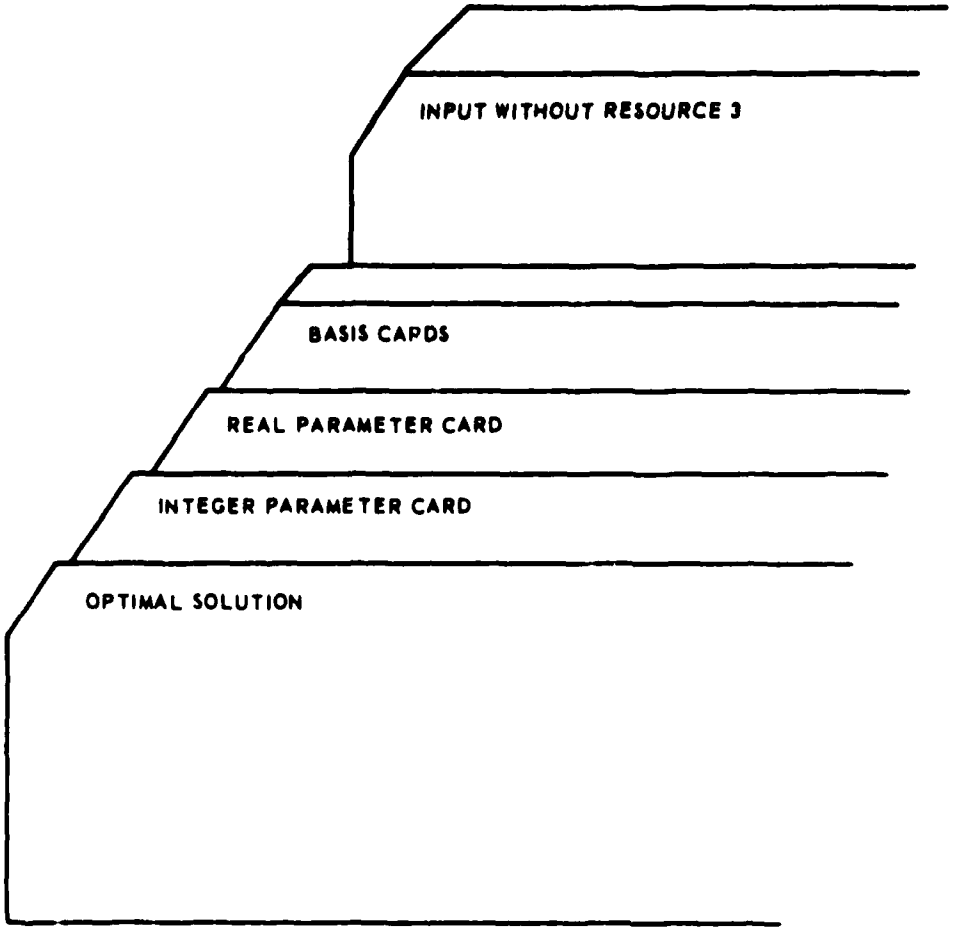


Fig. 4-11—Deck Structure for Main Program

of these groups are repeated for each node of the branching tree. The last group is for debugging purposes and consists of error messages (found in the appendix) and an error dump routine.

At the very beginning of the output there is a section which lists the values found on the parameter cards and the value of the bounds on the variables with concave functions. Following this is the general information about the branching tree which is output by the routine TABOUT. When called it outputs UO, the value of the best solution so far; USP, the value which all lower bounds must exceed before UO is considered optimal; and SIGMA, an array consisting of the right-hand-side values of the master variable constraints, and the lower bounds, upper bounds and cost function slopes of the master variables. The routine is called both before and after the LP is run. When called before the LP, it also prints the nodes on the branching tree which must still be considered. The information regarding these nodes includes the cost which represents a lower bound on that node, the number of the variable from which the next branch will be made, the value of that variable on which the branching will be performed, and the total fixed costs (E_k) associated with that node. In addition it checks to see if a new "best solution" has been found, and if it has, then it outputs that XZERO solution by column number and value.

The program is written such that there is a transformation from the actual problem to the linear problem which sets all of the lower bounds to zero. Thus, after the LP has derived a solution, this solution must be transformed back to a form applicable to the non-linear problem. When this is accomplished, this solution is output by the main routine, and it includes the actual cost of the solution PHI(XADJ), and the column numbers and their values for those columns in the basis.

RAC

The last form of output from the branching routines is a list of the master variables showing the differences between the linear estimates and the non-linear cost functions, and how these were calculated. It is the largest of these differences upon which the branching will be done.

Upon entry into the LP, a group of self-explanatory problem dimension items are output. These include; the number of standard rows in the matrix, the number of generalized upper bound rows, the number of logicals (slack variables) added to the problem, the number of columns in the matrix, the maximum number of columns allowed in core at any one time, the invert frequency or number of primal iterations before inverting the B-matrix, the maximum time the LP will be allowed to run, and the maximum number of primal iterations which may be executed by the LP. These last two parameters are set internally and are only used for debugging purposes.

During execution of the LP, the subroutine STATUS prints the status of the solution at each primal iteration. This first part of the output consists of the phase (1 - infeasible, 2 - feasible), the number of the iteration, a field called "try" which contains the number of iterations made with the current core columns plus 1000 times the maximum number before replenishment from the disc, the current value of the objective function, the number of potentially good columns in core (NDJS), and the number of artificial vectors still in the solution (NARTS). The second part of the output may contain a standard group of items of information, or it may contain a message giving a special condition which exists at that point. The standard output contains the DJ value of the column brought into the solution (a measure of the sensitivity of the objective function to that column), the internal column number of the column selected, where the internal number equals the actual number plus the number of

logicals, the status code* for that column, the internal column number of the column leaving the basis, its status code, and finally NSCAN, the number of columns in core plus 10000 times the number of disc reads performed. The special messages which may be given in place of this output are self-explanatory and have the general form; ....PRIMAL ... END OF PHASE 2 .. OPTIMAL, where PRIMAL is the routine sending the message.

There are a few other special messages which may appear in the output of the LP. After each inversion of the matrix, a check on the feasibility of the solution is made. If some column is not feasible, it is removed from the basis and a message is output of the form; INFEAS .. ROW xx COL xx VALUE xx BOUND xx. The last type of message has to do with degenerate columns. If it is determined that a column selected for entry to the basis will not change the solution, then it is rejected and another column is selected and the following message is output; REJECTED COL XX ROW xx PIVOT xx RHS xx.

Upon exit from the LP, MAPOUT will print the solution. It first gives the elements of the basis and their values, and then outputs the key variables in the GUB rows and their values. Finally it mixes and orders the basic columns and key columns and outputs the column numbers and values for the non-zero valued columns of the matrix.

Included in the output routines of the program is an error dump routine called ESCAPE. This routine outputs the arrays NAME - the status code, BASIS - columns in the basis, KEY - key columns in the GUB rows, JA - list of columns in core, JAREJ - the reject state of the columns in core (1 - rejected, 0 - otherwise), ALPHA - work space, BETA - solution vector of basic and key columns, GAMMA and DELTA - work spaces, and DJ - the objective function sensitivities to the columns in core.

RAC

THE REPORT GENERATOR

Program Logic

The logic structure of this program is not very complex. The first part works almost in reverse of the matrix generator. It searches the reference list to determine the symbolic names of the columns in the basis, and then decodes these symbolic names to properly account for the value of the variable. Most of the remaining portion of the program merely formats and prints the output.

Core Allocations

This program, like the matrix generator, stores all of its data in core. As a result almost three-fourths of core allocation is for common storage. Several of the dimensioning restrictions for the previous two programs are effective here also; namely, number of vehicles, number of period, and number of columns in the matrix. The only new limitation is that the number of years in the planning horizon be less than or equal to 20 (this does not include years from which vehicles have been inherited). The total core storage allocation is shown below:

		Octal	Decimal
CDC system routines	=	5,760	3,056
COMMON storage	=	51,554	21,356
REPGEN	=	6,176	3,198
SETUP	=	221	145
INSOLN	=	226	150
YRCOST	=	137	95
CINFO	=	307	199
PINFO	=	1,623	915
VALUES	=	73	59
TOTAL	=	70,765	29,173

User's Subroutine - YRCOST

This routine is, and must be, the same identical routine as is used by the matrix generator. One need only duplicate the routine and insert it in this program.

Input Formats

The input deck for this program can be constructed directly from the input deck to the matrix generator. It only uses a portion of the data but the formats are set up to use the same cards as were used previously; this is intended to help avoid errors in the preparation of input.

The first card is the title card, which is identical to that for the matrix generator. This is followed by the vehicle tables, which are placed in the order (numerical) in which they are listed on the output of the matrix generator. The vehicle tables each have the appropriate header card followed by card 1 and card type three (Fig. 4-6). Note that all cards of type 2 should be removed from the vehicle tables. Next come the period tables. These have header card and a card number 1 (Fig. 4-7). Only the cards number 1 are used in the period table. Finally the ENDTABLE card goes at the end of the deck, so that none of the task tables are used.

With this deck that has been extracted from the matrix generator, only one addition needs to be made before running. On the cards number 1 of the period tables, one must add in columns 11 and 12 the alphanumeric designator for the period. These designators are determined as follows; "00" for the period which contains the present year, "01" for the period which follows and starts with the first year of the planning horizon, "02" etc. for the periods sequentially that follows, "M1" for the period that precedes the present one, and "M2" etc. for the preceding period in

reverse chronological order. Since this field is read alphanumerically, it is essential that both columns be punched and the preceding zeros be included. The final deck has the general configuration shown in Fig. 4-12.

Output Description

For each solution that has been developed by the main program, the report generator will print four tables. The first contains the cost information and is fairly self-explanatory. It should be noted that the truncation savings is printed below the table, so the total cost in the table is the real cost and does not contain this credit. Also, although the total procurement cost is exact, the value during each period is a linear estimate to the nonlinear cost functions.

The other three tables give the resources purchased, stored, and used during each period. The only information that is not immediately available is the number of resources in the system, but this is merely the sum of the resources used plus the resources stored.

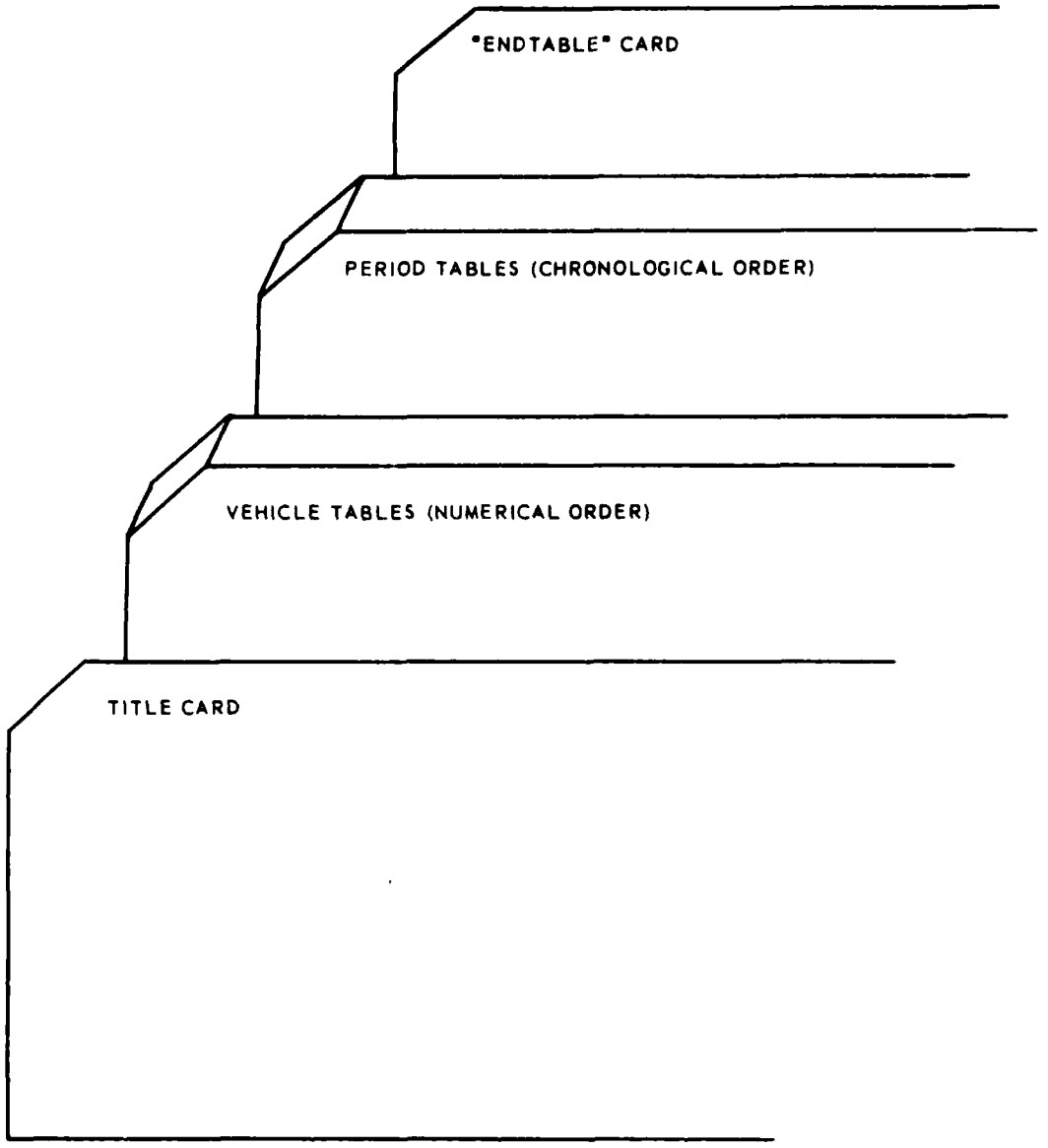


Fig. 4-12—Deck Structure for Report Generator

REFERENCES FOR VOLUME I

1. J. C. Hetrick, "Mathematical Models in Capital Budgeting," Chapter 7. New Decision Making Tools for Managers; edited by E. C. Bursk and J. F. Chapman, (Cambridge, Harvard University Press, 1963) p. 186.

2. J. E. Falk and R. M. Soland, "An Algorithm for Separable Non-Convex Programming Problems," Management Science, Vol. 15, No. 9, May 1969.

3. E. M. L., Beale, "Advanced Algorithmic Features for General Mathematical Programming Systems," Integer and Nonlinear Programming, edited by J. Abadie, North-Holand Publishing Company, 1970.

REFERENCES FOR VOLUME II

4. R. New, "Optimal Planning Over Time — OPT," Journal of Systems Management, March 1972.

5. R. M. Soland, "An Algorithm For Separable Non-Convex Programming Problems II: Non-Convex Constraints," Management Science, Volume 17, No. 11, July 1971.